



Projekt Systemu

Projekt i implementacja aplikacji
do zarządzania wypożyczalnią samochodów

Przemysław Rutkowski,

Łukasz Seremak

grupa III

Informatyka (niestacjonarna), semester VI

Dokumentacja.md

Pliki źródłowe niniejszej dokumentacji w formacie Markdown są dostępne pod poniższym adresem:

<https://github.com/lukaszse/car-rental/blob/master/dokumentacja/Dokumentacja.md>

Działająca wersja MVP aplikacji dostępna jest pod adresem:

<https://ubuntu.lseremak.p3.tiktalik.io/car-rental/home>

1. Cel projektu

Celem projektu jest zaprojektowanie oraz implementacja aplikacji webowej (w wersji MVP) wspomagającej procesy obsługi zleceń związanych z wypożyczaniem samochodów osobowych dla klientów indywidualnych. Poprzez informatyzację wszystkich procesów obsługi klienta aplikacja zapewni szereg korzyści m.in.:

- redukcję kosztów obsługi zamówień poprzez ich całkowitą automatyzację,
- zwiększenie wydajności obsługi klientów,
- zapewnienie bezpieczeństwa danych przechowywanych w scentralizowanej bazie danych.

2. Słownik pojęć

- **MTFB** — (ang. Mean Time between Failures) średni czas pomiędzy wystąpieniem awarii¹.
- **Docker** — otwarte oprogramowanie do wirtualizacji, umożliwiające "konteneryzację" tj. pozwalające umieścić program oraz jego zależności (biblioteki) w przenośnym wirtualnym kontenerze który można uruchomić na dowolnym serwerze z systemem Linux, Windows i MacOS².
- **GUI** — (ang. Graphical User Interface) graficzny interfejs użytkownika.
- **RODO** — rozporządzenie o ochronie danych osobowych — rozporządzenie unijne, zawierające przepisy o ochronie osób fizycznych w związku z przetwarzaniem danych osobowych oraz przepisy o swobodnym przepływie danych osobowych³.
- **PDF** — (and. Portable Document Format) format plików służący do prezentacji, przenoszenia i drukowania treści tekstowo-graficznych, stworzony przez firmę Adobe Systems. Obecnie rozwijany i utrzymywany przez Międzynarodową Organizację Normalizacyjną⁴.

- **Spring Boot** — framework do budowania aplikacji, w tym aplikacji webowych w języku java⁵.
- **Thymeleaf** — silnik szablonów html⁶.
- **H2** — baza danych SQL przechowująca dane w pliku lub w pamięci operacyjnej, stosowana do testów lub prostych aplikacji⁷.
- **Spock** — framework do tworzenia testów jednostkowych, wykorzystujący język Groovy⁸.
- **MVP** — (ang. Minimum Viable Product) - produkt o minimalnej funkcjonalności potrzebnej do wprowadzenia na rynek⁹.

¹ <https://pl.wikipedia.org/wiki/MTBF>

² <https://www.docker.com/>

³ https://pl.wikipedia.org/wiki/Og%C3%B3lne_rozporz%C4%85dzenie_o_ochronie_danych

⁴ https://pl.wikipedia.org/wiki/Portable_Document_Format

⁵ <https://spring.io/projects/spring-boot>

⁶ <https://www.thymeleaf.org/>

⁷ <https://www.h2database.com/>

⁸ <https://spockframework.org/>

⁹ <https://www.biznesowerewolucje.com/mvp-minimum-viable-product-praktycznie/>

3. Szczegółowy opis wymagań

3.1. Wymagania funkcjonalne do zaimplementowania w wersji MVP

System umożliwia:

- wyszukiwanie dostępnych w określonym terminie samochodów, wg zadanych kryteriów takich jak:
 - marka,
 - model;
- wyświetlenie szczegółowych informacji na temat wybranego pojazdu;
- rejestrację użytkowników;
- logowanie użytkowników;
- zarządzanie użytkownikami w trybie *Administratora* (dodawanie/edycja/usuwanie);
- przeglądanie pojazdów w trybie *Gościa*;
- dokonanie rezerwacji przez zarejestrowanego i zalogowanego użytkownika;
- odwołanie rezerwacji przez osobę zarządzającą;

- rejestrację użytkowników oraz modyfikację danych przez użytkowników;
- przeglądanie własnych rezerwacji;
- dodawanie/usuwanie oraz modyfikacje pojazdów przez osobę zarządzającą;
- przeglądanie listy zarezerwowanych oraz wypożyczonych samochodów przez osobę zarządzającą;
- obsługę płatności;
- generowanie i pobieranie potwierdzenia rezerwacji w formacie pdf;
- kontaktowanie się z obsługą wypożyczalni poprzez formularz kontaktowy;
- wysyłanie wiadomości do obsługi serwisu;
- wysyłanie wiadomości do obsługi w trybie *Gościa* zabezpieczone reCaptcha v2;
- walidacja dla wszystkich wprowadzonych pól wraz z systemem alertów/ostrzeżeń o źle wprowadzonych danych (pola o szczególnej składni jak kod pocztowy czy email walidowane z wykorzystaniem wyrażeń regularnych);
- zabezpieczenie ścieżek URL (dostęp do ścieżki tylko dla użytkowników uprawnionych);
- przypisywanie ról dla użytkowników przez *Administradora*;
- Bezpieczeństwo danych użytkownika np. poprzez wykorzystanie protokołu HTTPS;

3.2. Wymagania funkcjonalne do zaimplementowania w przyszłych wersjach oprogramowania

- rozszerzenie wyszukiwania dostępnych samochodów o kryteria takie jak:
 - rodzaj skrzyni biegów,
 - rodzaj silnika (benzyna/diesel/elektryczny);
- zmiana statusu z rezerwacji na wypożyczenie;
- generowanie faktur dla rezerwacji;
- generowanie korekty faktury w przypadku odwołania zlecenia.

3.3. Wymaganie niefunkcjonalne

- GUI:
 - aplikacja webowa z interfejsem dla przeglądarki internetowej,
 - spójny wygląd zgodnie z zaakceptowanym szablonem (spójna kolorystyka, menu, zachowanie się systemu).
- Dostępność:
 - obsługa języków: polski,
 - obsługa przeglądarek: Chrome, Safari, Edge.
- Niezawodność:
 - System dostępny 24/7. MTFB = 1000h.
- Bezpieczeństwo:

- aplikacja jest uruchamiana tylko i wyłącznie używając protokołu https, który zapobiega przechwytywaniu i zmienianiu przesyłanych danych. A dodatkowo zaimplementowane zostało:
 - haszowanie haseł,
 - automatyczne wylogowanie użytkownika po upływie 10 minut,
 - spełnia wymagania Ustawy z dnia 10 maja 2018 r. o ochronie danych osobowych (RODO).
- Tabele danych do wyświetlenia, w szczególności dla danych filtrowanych ładowane bez przeładowywania całej strony. Architektura typu Single Page Application nie jest wymagana, dopuszczalna jest architektura hybrydowa.

3.4. Ograniczenia

- System musi być instalowany z obrazu Dockera pobieranego online¹.
- Wszystkie podstrony aplikacji działające wyłącznie przez HTTPS
- System musi być zgodny z ustawą o ochronie danych osobowych RODO.
- System musi obsługiwać przeglądarki Chrome i Edge.
- MVC zbudowane na podstawie plikową bazę danych zintegrowaną z aplikacją, aby wyeliminować konieczność tworzenia osobnej bazy danych.

¹ W celu uproszczenia wdrożenia aplikacji, autorzy zdecydowali się zastosować konteneryzację, a co za tym idzie udostępnić aplikację jak obraz Dockera.

3.5. Architektura aplikacji

3.5.1. Aplikacja MVC

Aplikacja wykorzystywać będzie wzorzec projektowy MVC. Zgodnie ze wzorcem MVC będzie podzielona na 3 moduły:

- Model reprezentujący dane (np. pobierane z bazy danych czy parsowane z plików XML)
- Widok reprezentujący interfejs użytkownika
- Kontroler, czyli logikę sterującą aplikacją Logika sterująca kontrolera, ze względu na poprawienie czytelności kodu, będzie rozdzielona na kontroler, który będzie odpowiedzialny za obsługę zapytań zewnętrznych, oraz serwis odpowiedzialny za realizację logiki biznesowej, oraz będący łącznikiem z warstwą modelu, zapewniającą dostęp do bazy danych. Taki podział będzie dotyczył się każdego z poszczególnych widoków aplikacji.

3.5.2. Aplikacja monolityczna / hybrydowa

Aplikacja zostanie zbudowana jako aplikacja monolityczna. Jednakże budowa systemu umożliwia przyszłą modularyzację, poprzez dołączenie dodatkowych mikroserwisów, które byłyby odpowiedzialne za nowe grupy funkcji.

3.5.3. Architektura warstwy klienckiej (frontend)

Frontend aplikacji zostanie stworzony z wykorzystaniem silnika szablonów Thymeleaf wspieranego przez Spring Boot. Oprócz statycznych szablonów warstwa frontend będzie wyposażona w dynamiczne elementy obsługiwane za pomocą JavaScript (AJAX), które zostaną wykorzystane m.in. do budowania dynamicznych tablic pozwalających wspierających mechanizm wyszukiwania (filtrowania rekordów).

4. Użytkownicy (Aktorzy/Role)

4.1 Uprawnienia użytkowników

W systemie funkcjonować będą użytkownicy o następujących rolach:

1. *Administrator* (ang. Admin¹)

- posiada uprawnienia wszystkich użytkowników, a ponadto ma możliwość zarządzania *Użytkownikami* oraz przypisywania użytkownikom określonej roli.

2. *Zarządca* (ang. Manager¹)

- posiada uprawnienia *Użytkownika* oraz *Gościa* a ponadto:
 - posiada możliwość przeglądania wszystkich rezerwacji,
 - posiada możliwość anulowania dowolnej rezerwacji (swojej lub innego użytkownika),
 - posiada możliwość modyfikacji dowolnej rezerwacji (w imieniu siebie lub innego użytkownika),
 - posiada możliwość złożenia rezerwacji za *Użytkownika* (np. rezerwacja telefoniczna).

3. *Użytkownik* (ang. User¹)

- posiada uprawnienia *Gościa*, a ponadto:
 - posiada możliwość dokonywania rezerwacji,
 - posiada możliwość opłacania rezerwacji,
 - posiada możliwość przeglądania swoich rezerwacji,
 - posiada możliwość generowania faktur².

4. *Gość* (ang. Guest¹)

- posiada możliwość wyszukiwania pojazdów w wybranym terminie, spełniających wybrane kryteria.

¹ W związku z przewidywaną dwujęzycznością projektu, w diagramach dokumentacji zostały użyte określenia w stosunku do ról zarówno w języku polskim, jak i angielskim.

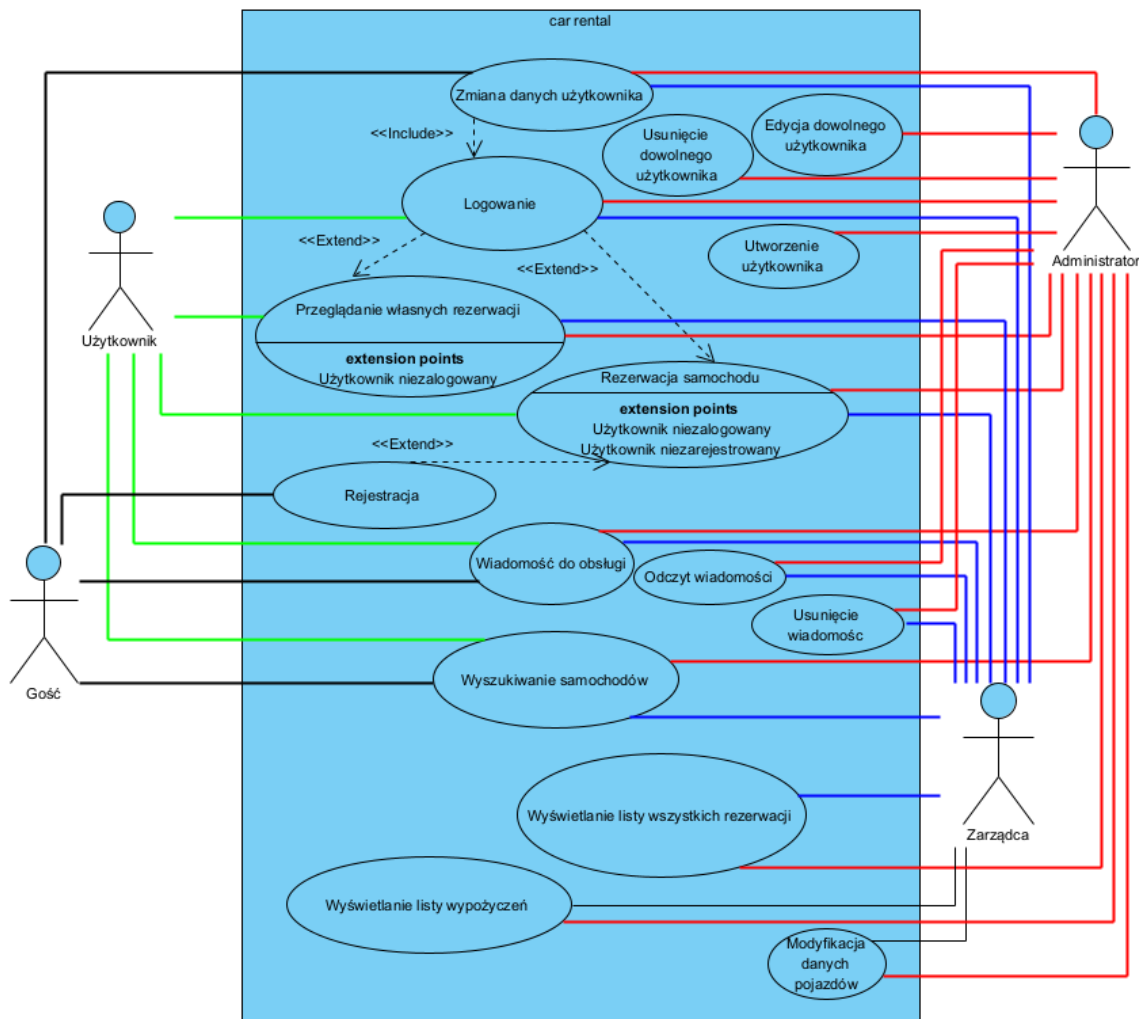
² Opcja generowania faktur nie jest zaimplementowana w wersji MVP. Będzie zaimplementowana w kolejnych wersjach aplikacji.

4.2. Tworzenie użytkowników różnego typu

Po zainstalowaniu aplikacji w systemie będą stworzone trzy podstawowe konta użytkowników (*Adminisrator*, *Użytkownik*, *Zarządca*). Po zainstalowaniu należy zmienić standardowe hasła użytkowników. Należy mieć na względzie, że zarządzanie użytkownikami możliwe jest tylko z poziomu administratora. W tym celu należy wykorzystać konto stworzone automatycznie podczas instalacji. Z tego konta można utworzyć dodatkowe konta wszystkich typów.

Podczas rejestracji konta przez *Gościa* stworzone zostanie konto *Użytkownika*. Ewentualna zmiana roli możliwa jest wyłącznie przez konto administratora.

5. Diagramy przypadków użycia (wybrane przykłady)



6. Scenariusze przypadków użycia

Poniżej przedstawiono wybrane scenariusze przypadków użycia spośród wszystkich scenariuszy:

- logowanie do systemu,
- rejestracja użytkownika,
- wysłanie wiadomości do obsługi,
- odczyt wiadomości przez *Administradora/Zarządcę*,
- usuwanie wiadomości przez *Administradora/Zarządcę*,
- wyszukiwanie samochodów,
- rezerwacja samochodu,
- wyświetlanie listy wszystkich rezerwacji przez *Administradora/Zarządcę*,
- przeglądanie własnych rezerwacji przez użytkownika,
- zmiana danych własnych użytkownika,
- tworzenie użytkownika przez *Administradora*,

- edycja dowolnego użytkownika przez *Administradora*,
- usuwanie dowolnego użytkownika przez *Administradora*,
- wyświetlanie listy wypożyczeń.

6.1. Logowanie do systemu

Nazwa przypadku użycia	Zaloguj się do systemu	
Cel w kontekście systemu	Zarejestrowany użytkownik loguje się do system.	
Warunki wstępne	Użytkownik posiada konto w systemie.	
Warunek pomyślnego zakończenia	Użytkownik pomyślnie zalogował się do systemu.	
Stan końcowy - niepowodzenie	Zwrócono informację o nieprawidłowym loginie i/ lub hasle.	
Główni Aktorzy	Użytkownik, Zarządca, Administrator	
Aktorzy współuczestniczący	-	
Wywołanie (incjacja) przypadku użycia	Użytkownik żąda od systemu możliwości zalogowania się (otwarcie formularza logowania)	
Przypadki użycia - include	-	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Użytkownik żąda od system możliwości zalogowania się (otwarcie formularza logowania)
	2	Użytkownik wpisuje login i hasło.
	3	Dane są weryfikowane przy pomocy danych pobranych z wewnętrznej bazy danych aplikacji z użytkownikami.
	4	Użytkownik zostaje zalogowany do systemu.
	5	Użytkownik zostaje przekierowany do menu dla zarejestrowanych użytkowników.
Rozszerzenia głównego przepływu	Step	Branching Action
	3.1	Informacje uzyskane z bazy danych użytkowników nie pozwalają na potwierdzenie danych użytkownika.
	3.2	Próba zalogowania zostaje odrzucona.

6.2. Zmiana danych

Nazwa przypadku użycia	Zmień dane użytkownika	
Cel w kontekście systemu	Zarejestrowany użytkownik zmienia swoje dane w systemie.	
Warunki wstępne	Użytkownik posiada konto w systemie.	
Warunek pomyślnego zakończenia	Użytkownik pomyślnie zmienił swoje dane w systemie.	
Stan końcowy - niepowodzenie	Zmiana danych użytkownika została odrzucona.	
Główni Aktorzy	Użytkownik, Zarządca, Administrator	
Aktorzy współuczestniczący	-	
Wywołanie (incjacja) przypadku użycia	Użytkownik żąda od systemu zmiany swoich danych.	
Przypadki użycia - include	Zaloguj się do systemu	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Użytkownik żąda od system zmiany swoich danych.
	2	Użytkownik loguje się do systemu Include::Zaloguj się do systemu
	3	Użytkownik zmienia dane w systemie
	4	Nowe dane są walidowane
	5	Dane w systemie zostają zmienione
Rozszerzenia głównego przepływu	Step	Branching Action
	2.1	Użytkownik nie ma prawa zmienić danych
	2.2	Wniosek o zmianę danych jest odrzucany
	4.1	Wynik walidacji wprowadzonych przez użytkownika danych jest negatywny
	4.2	Próba zmiany danych zostaje odrzucona

6.3. Rezerwacja samochodu

Nazwa przypadku użycia	Zarezerwuj samochód	
Cel w kontekście systemu	Zarejestrowany użytkownik rezerwuje samochód	
Warunki wstępne	Użytkownik jest zalogowany do systemu, Na ekranie wyświetlany jest dostępny samochód.	
Warunek pomyślnego zakończenia	Użytkownik pomyślnie zarezerwował dostępny samochód	
Stan końcowy - niepowodzenie	Rezerwacja pojazdu została odrzucona	
Główni Aktorzy	Użytkownik, Zarządca, Administrator	
Aktorzy współuczestniczący	-	
Wywołanie (incjacja) przypadku użycia	Użytkownik żąda zarezerwowania wybranego samochodu w systemie	
Przypadki użycia - include	-	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Użytkownik żąda zarezerwowania dostępnego samochodu na określony czas
	2	Dostępność samochodu jest weryfikowana przy pomocy danych z bazy danych [kalendarz]
	3	Samochód zostaje zarezerwowany. W bazie danych ustawiany jest okres wypożyczenia
	4	Użytkownik zostaje poinformowany o dokonanej rezerwacji oraz zostaje przekierowany do systemu płatności
	5	E-mail z informacją o dokonanej rezerwacji zostaje przesłany do klienta
Rozszerzenia głównego przepływu	Step	Branching Action
	2.1	Z danych z bazy pojazdów wynika że samochód jest niedostępny w wybranym terminie
	2.2	Próba rezerwacji pojazdu zostaje odrzucona, z powodu niedostępności pojazdu w wybranym terminie

6.4. Usuwanie wiadomości

Nazwa przypadku użycia	Usuwanie wiadomości	
Cel w kontekście systemu	Administrator lub Zarządca usuwa wiadomości od użytkowników	
Warunki wstępne	Zalogowany użytkownik ma uprawnienia administratora lub zarządcy, na ekranie wyświetlany jest panel z wiadomościami	
Warunek pomyślnego zakończenia	Wiadomość została usunięta	
Stan końcowy - niepowodzenie	Wiadomość nie została usunięta	
Główni aktorzy	Zarządca, Administrator	
Aktorzy współuczestniczący	-	
Wyzwalacz	Zarządca/Administrator żąda od systemu usunięcia wiadomości w systemie	
Przypadki użycia - include	-	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Użytkownik żąda od systemu możliwości usunięcia wiadomości
	2	System weryfikuje polecenie w bazie danych
	3	Wiadomość zostaje usunięta
Rozszerzenia głównego przepływu	Step	Branching Action
	3.1	Próba usunięcia wiadomości zostaje odrzucona

6.5. Edycja danych dowolnego użytkownika

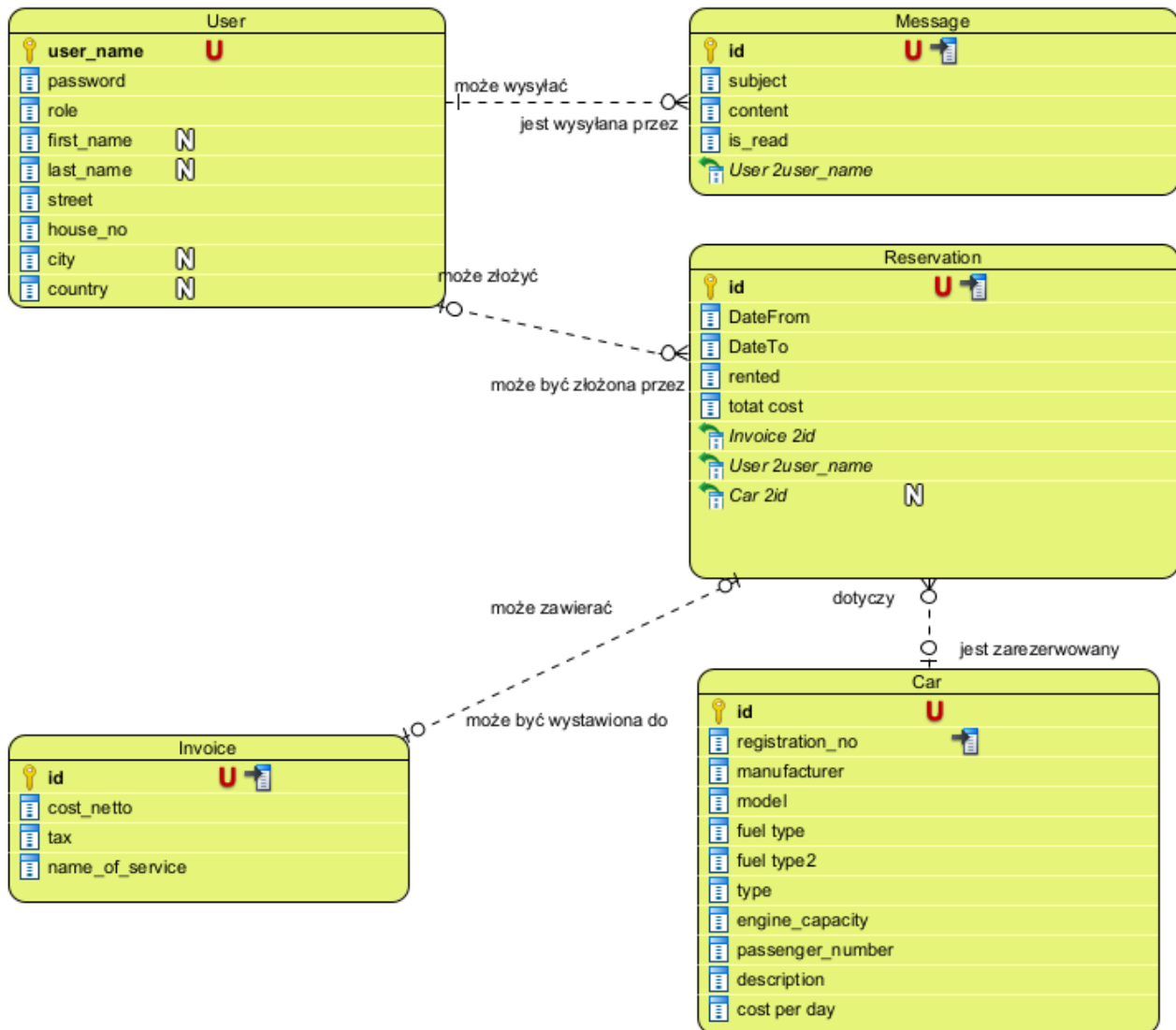
Nazwa przypadku użycia	Edycja danych dowolnego użytkownika	
Cel w kontekście systemu	Administrator edytuje dane dowolnego użytkownika	
Warunki wstępne	Zalogowany użytkownik ma uprawnienia administratora, na ekranie jest wyświetlany panel edycji użytkowników	
Warunek pomyślnego zakończenia	Administrator pomyślnie zmienił dane użytkownika w systemie	
Stan końcowy - niepowodzenie	Zmiana danych została odrzucona	
Główni aktorzy	Administrator	
Aktorzy współuczestniczący	-	
Wyzwalacz	Administrator żąda zmiany danych wybranego użytkownika	
Przypadki użycia - include	-	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Administrator żąda od systemu możliwości zmiany danych użytkownika
	2	Administrator zmienia dane w systemie
	3	Nowe dane są walidowane
	4	Dane w systemie zostają zmienione
Rozszerzenia głównego przepływu	Step	Branching Action
	3.1	Walidacja wprowadzonych danych daje wynik negatywny
	3.2	Próba zmiany danych zostaje odrzucona

6.6. Dodanie nowego użytkownika

Nazwa przypadku użycia	Dodanie nowego użytkownika	
Cel w kontekście systemu	Administrator dodaje ręcznie użytkownika	
Warunki wstępne	Zalogowany użytkownik ma uprawnienia administratora, na ekranie jest wyświetlany panel edycji użytkowników	
Warunek pomyślnego zakończenia	Administrator pomyślnie dodał użytkownika do systemu	
Stan końcowy - niepowodzenie	Wniosek o dodanie nowego użytkownika został odrzucony	
Główni aktorzy	Administrator	
Aktorzy współuczestniczący	-	
Wyzwalacz	Administrator dodaje nowego użytkownika	
Przypadki użycia - include	-	
Przypadki użycia - extend	-	
Główny przepływ	Step	Action
	1	Administrator żąda od systemu możliwości dodania nowego użytkownika
	2	Administrator uzupełnia formularz z danymi nowego użytkownika
	3	Nowe dane są walidowane
	4	Użytkownik zostaje dodany
Rozszerzenia głównego przepływu	Step	Branching Action
	3.1	Walidacja wprowadzonych danych daje wynik negatywny
	3.2	Próba dodania użytkownika zostaje odrzucona

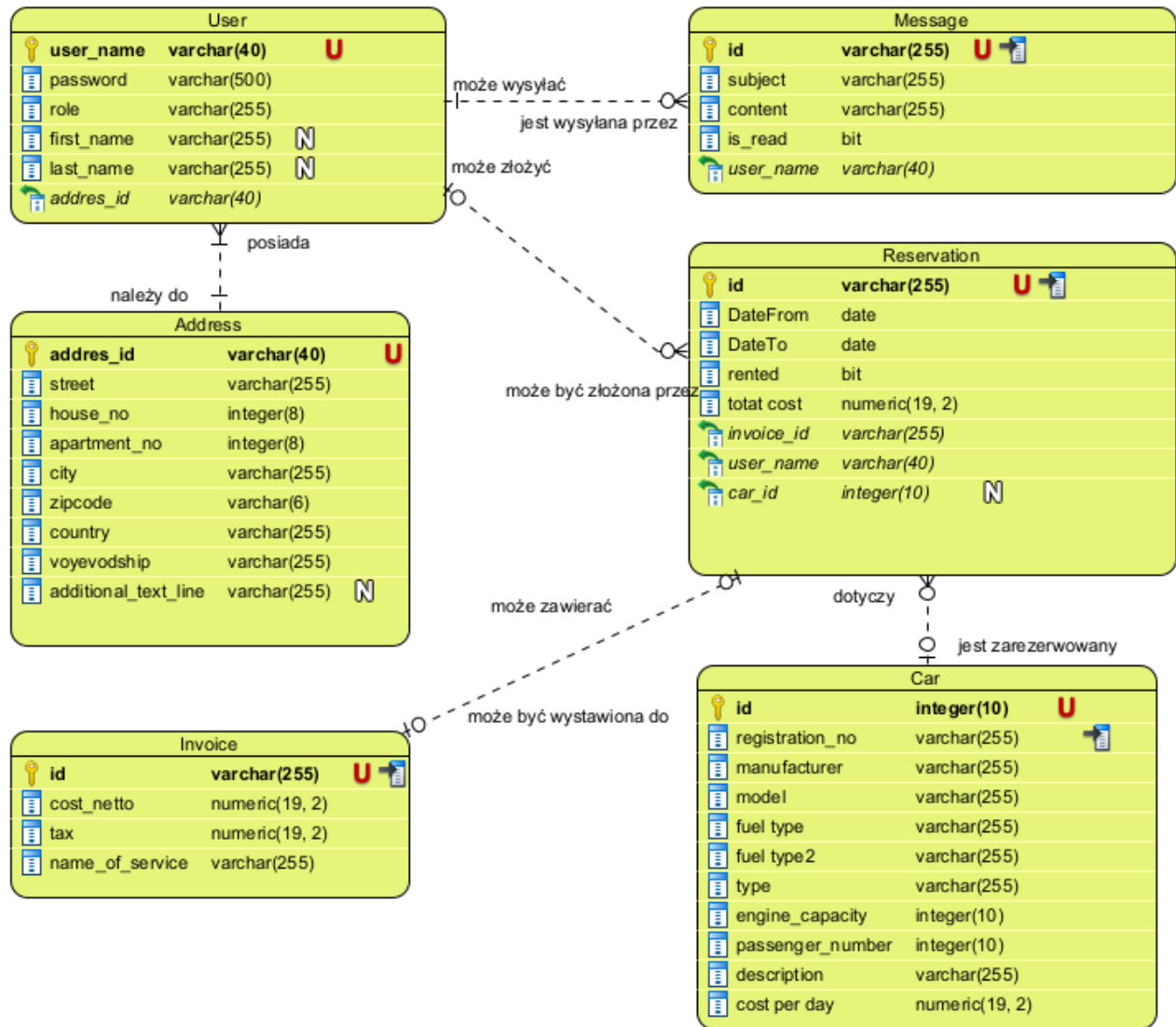
7. Model bazy danych

7.1. Model konceptualny

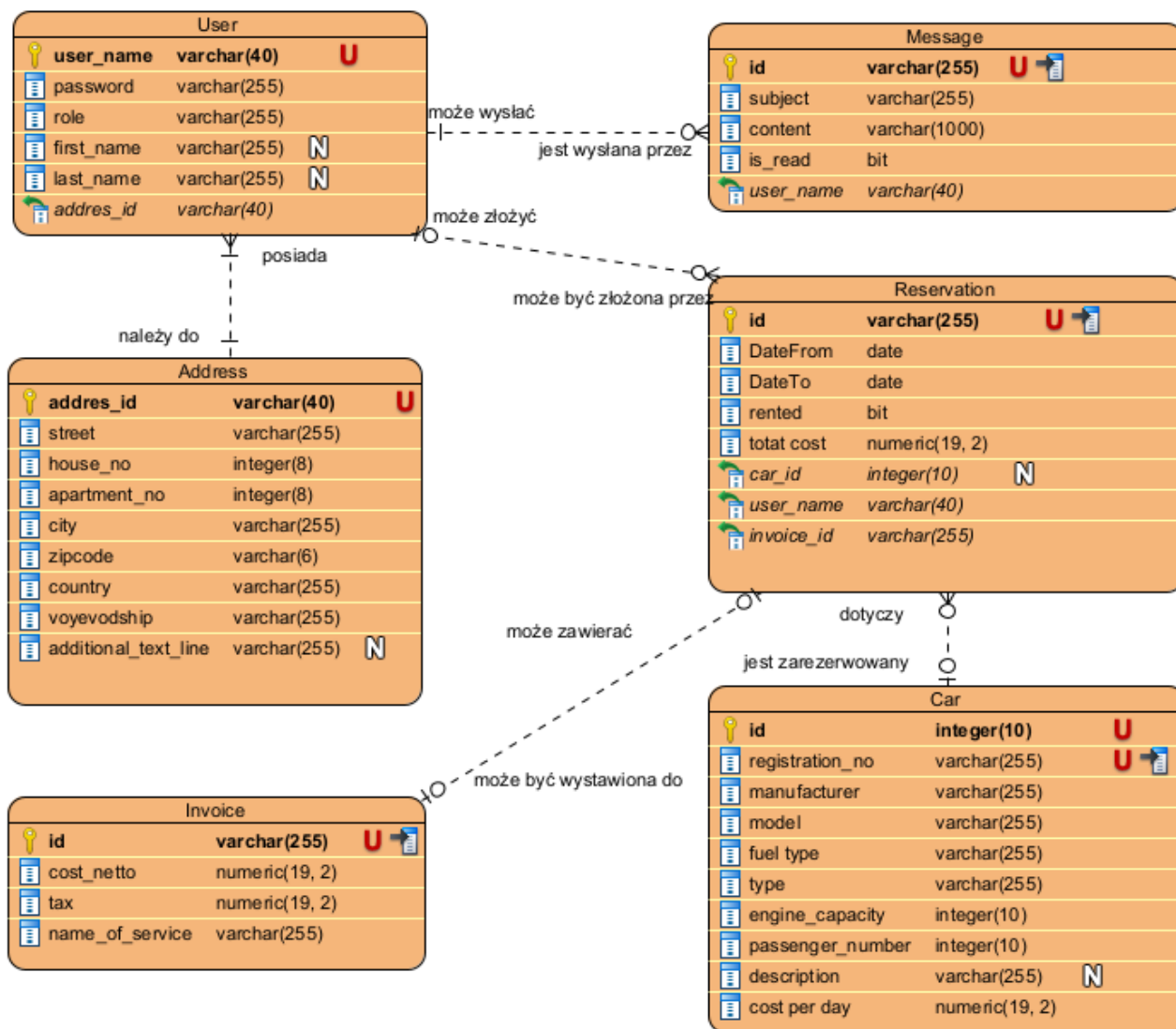


7.2. Model logiczny (ERD)

W ramach normalizacji bazy danych została stworzona tabela `Address`. Tabele `Address` oraz `Invoice` ta nie występują w wersji MVP aplikacji. Zostania one zaimplementowana w kolejnych wersjach.



7.3. Model relacyjny (Fizyczny)



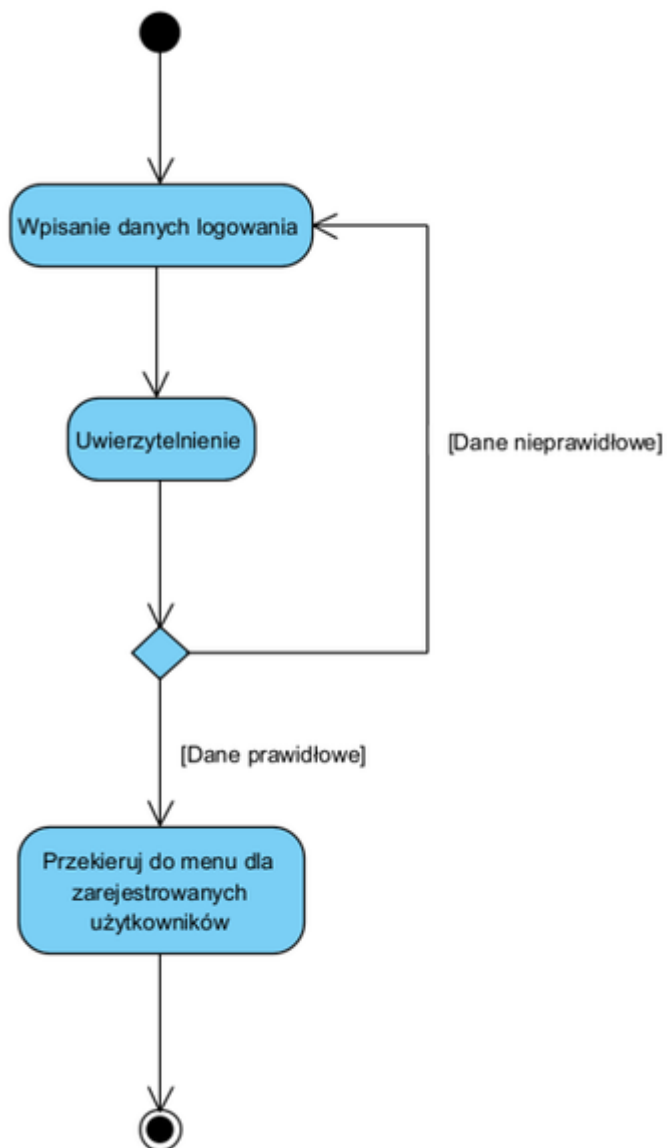
8. Diagramy czynności (wybrane przykłady)

Poniżej przedstawiono wybrane diagramy czynności. Wszystkie diagramy znajdują się na poniższej liście:

- logowanie do systemu,
- rejestracja użytkownika,
- wysłanie wiadomości do obsługi,
- odczyt wiadomości przez *Administradora/Zarządcę*,
- usuwanie wiadomości przez *Administradora/Zarządcę*,
- wyszukiwanie samochodów,
- rezerwacja samochodu,
- wyświetlanie listy wszystkich rezerwacji przez *Administradora/Zarządcę*,
- przeglądanie własnych rezerwacji przez użytkownika,
- zmiana danych własnych użytkownika,
- tworzenie użytkownika przez *Administradora*,

- edycja dowolnego użytkownika przez *Administradora*,
- usuwanie dowolnego użytkownika przez *Administradora*,
- wyświetlanie listy wypożyczeń.

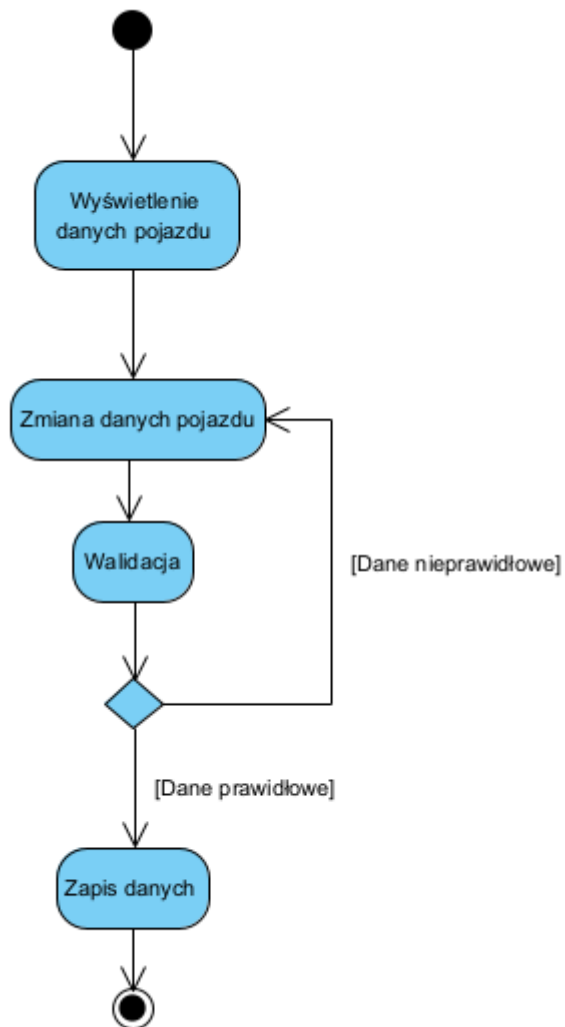
8.1. Logowanie do systemu



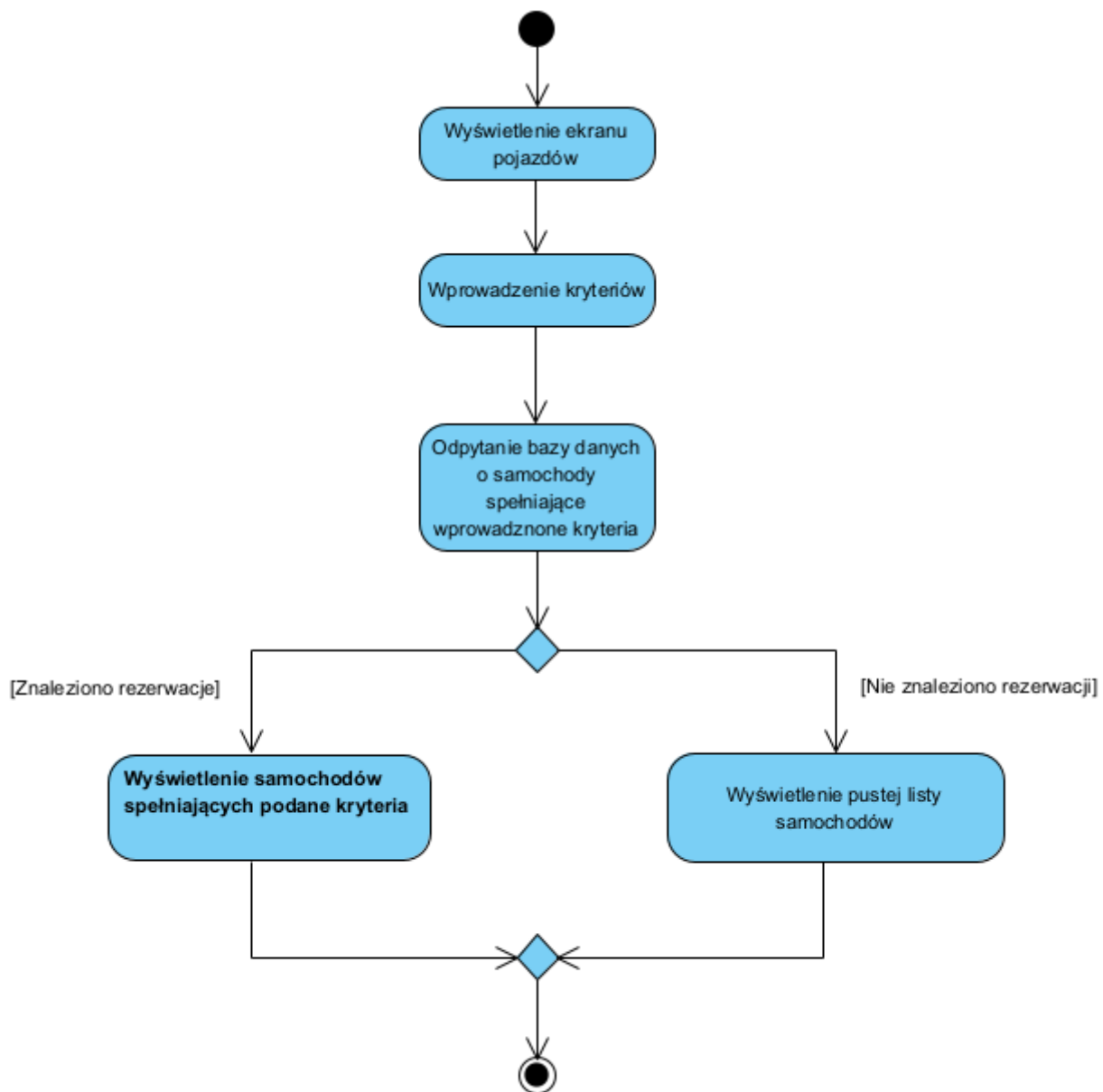
8.2. Zmień dane użytkownika



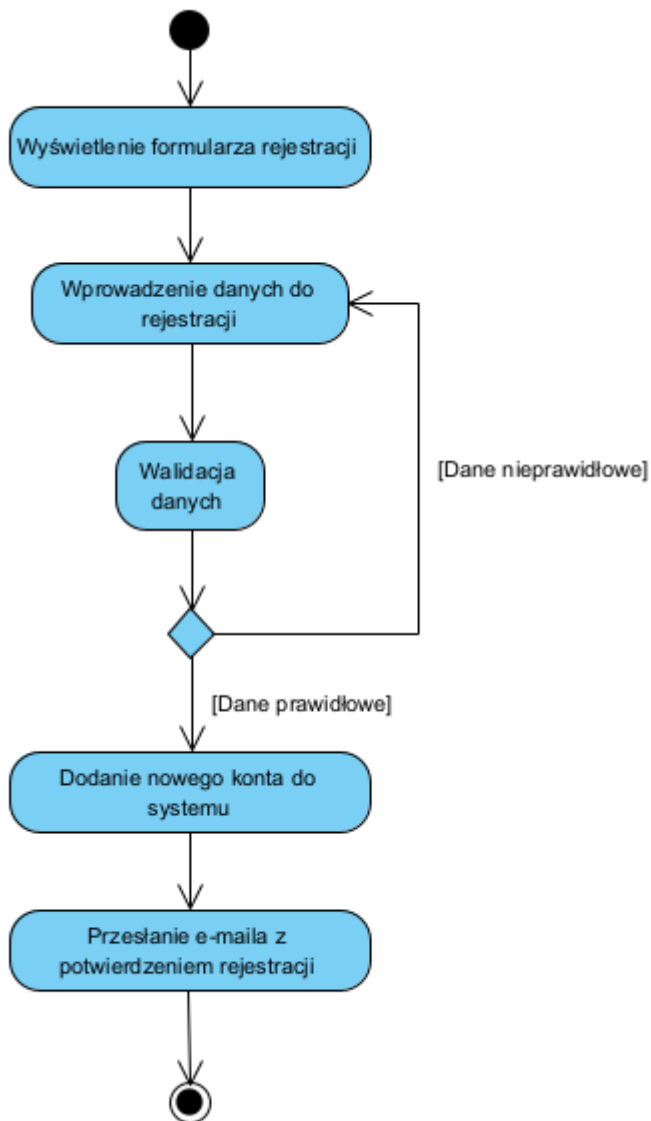
8.3. Zmiana danych pojazdu (tylko *Administrator/Zarządca*)



8.4 Wyszukiwanie samochodów



8.5. Rejestracja użytkownika



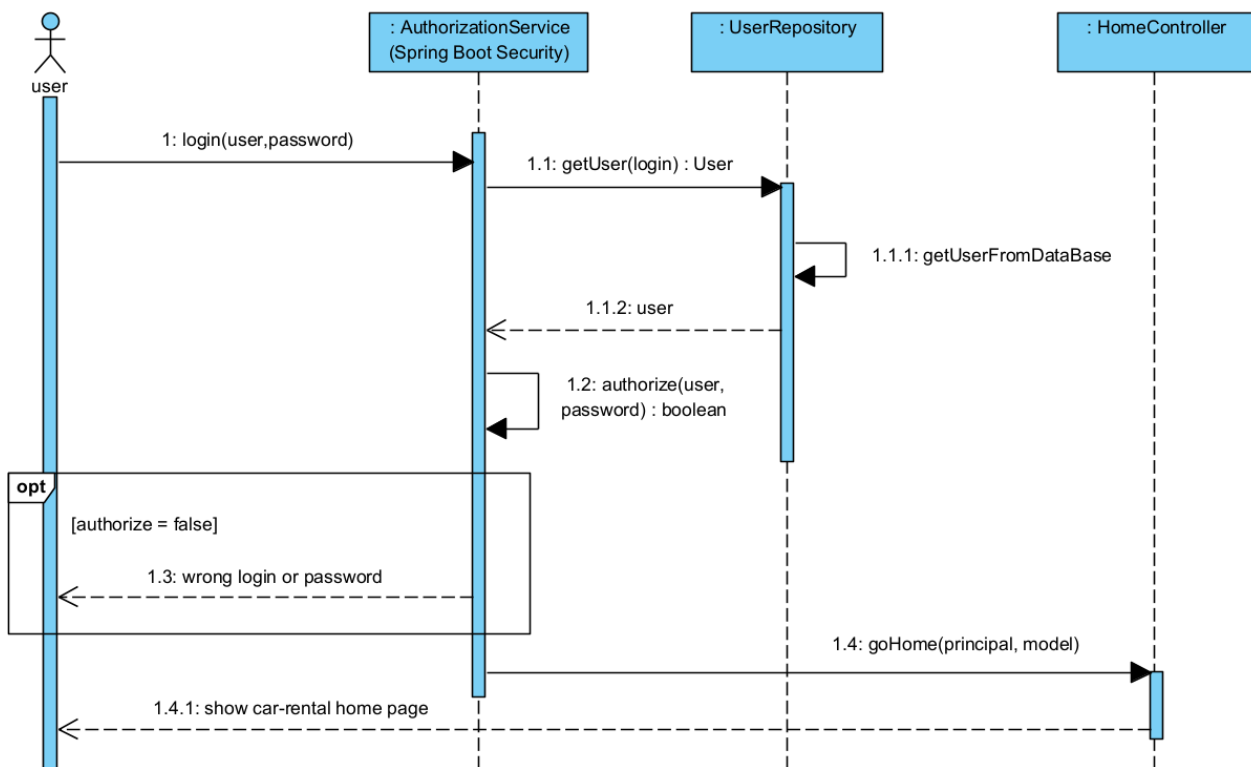
9. Diagramy sekwencji (wybrane przykłady)

Poniżej przedstawiono wybrane diagramy sekwencji. Wszystkie diagramy znajdują się na poniższej liście:

- logowanie do systemu,
- rejestracja użytkownika,
- wysłanie wiadomości do obsługi,
- odczyt wiadomości przez *Administradora/Zarządcę*,
- usuwanie wiadomości przez *Administradora/Zarządcę*,
- wyszukiwanie samochodów,
- rezerwacja samochodu,
- wyświetlanie listy wszystkich rezerwacji przez *Administradora/Zarządcę*,
- przeglądanie własnych rezerwacji przez użytkownika,
- zmiana danych własnych użytkownika,

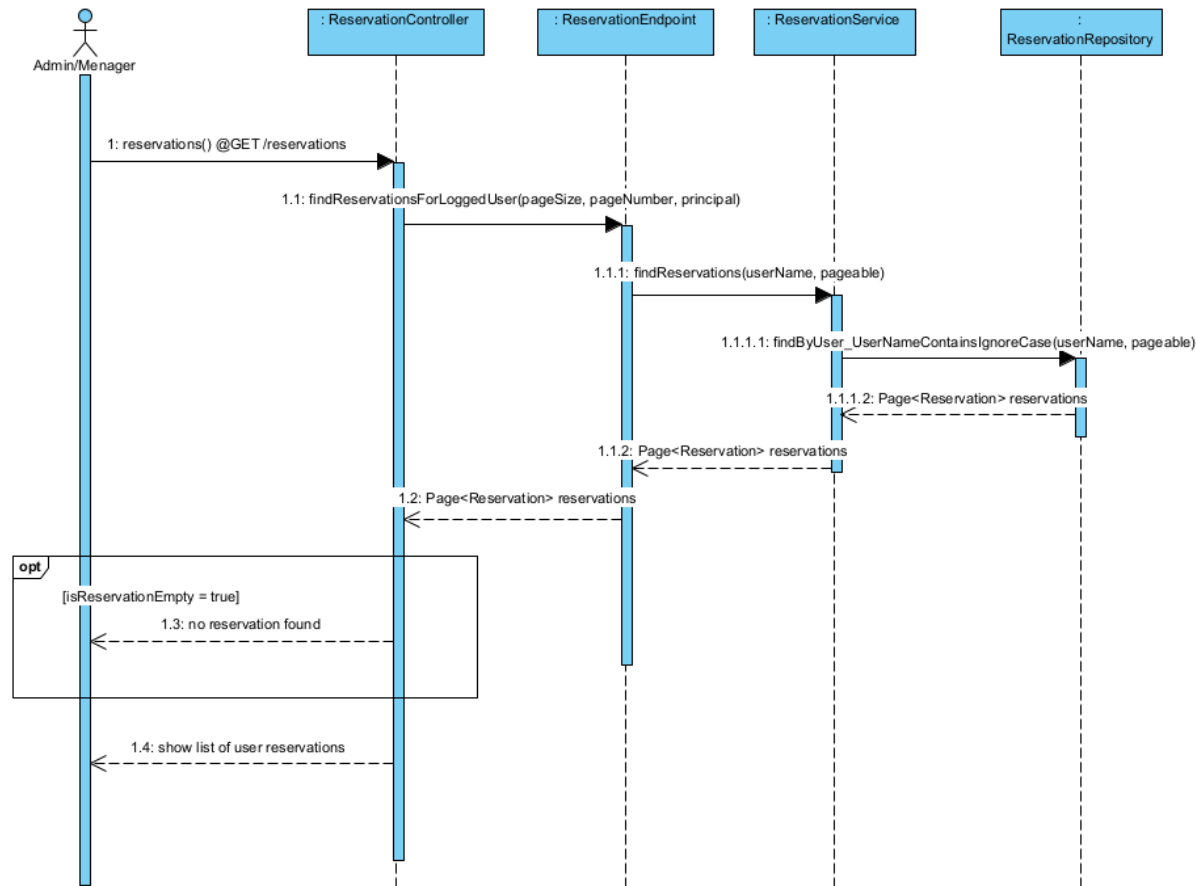
- tworzenie użytkownika przez *Administradora*,
- edycja dowolnego użytkownika przez *Administradora*,
- usuwanie dowolnego użytkownika przez *Administradora*,
- wyświetlanie listy wypożyczeń.

9.1. Logowanie do systemu

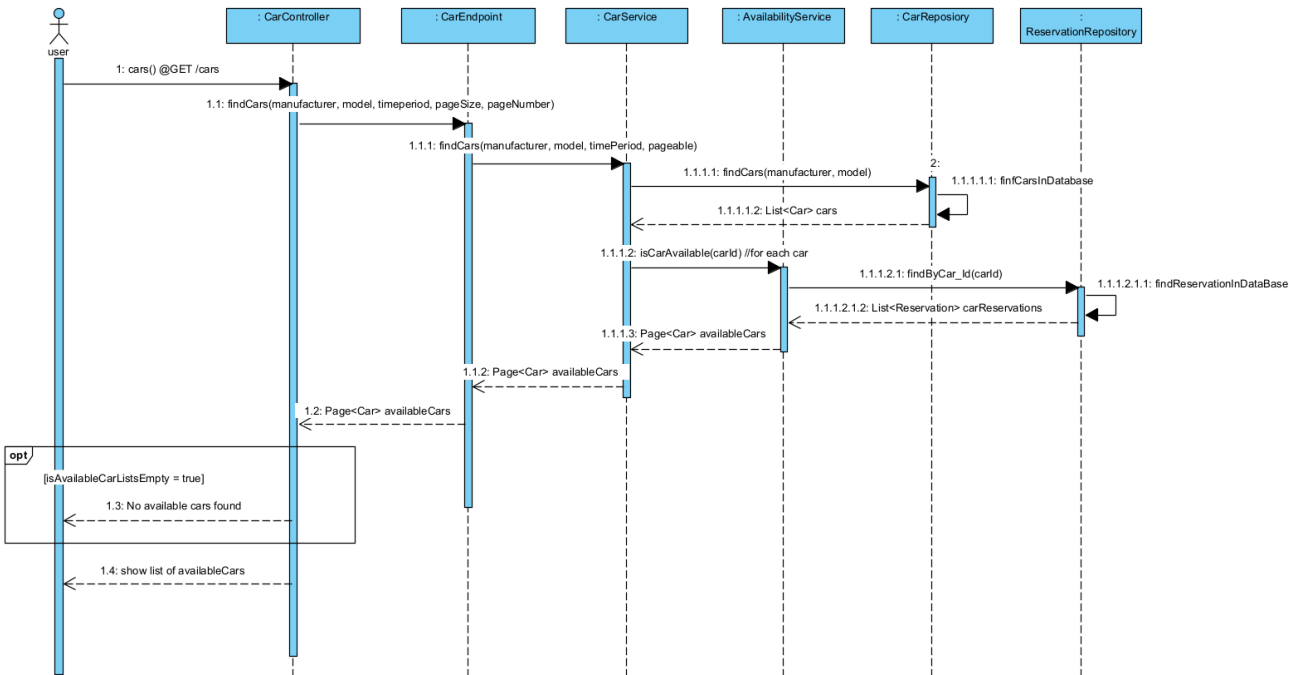


Uwaga Klasa `AuthorizationService` to klasa należąca do Spring Boot Security, tj. zestawu bibliotek pozwalających na obsługę funkcji bezpieczeństwa aplikacji.

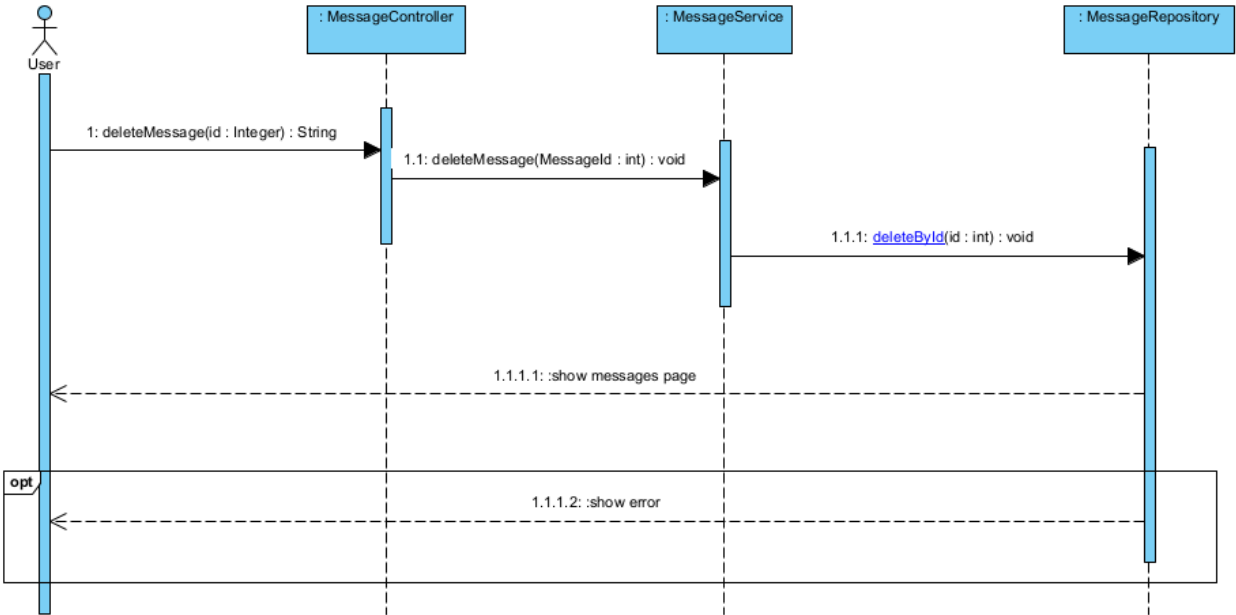
9.2. Wyświetlanie listy wszystkich rezerwacji przez *Administradora/Zarządcę*



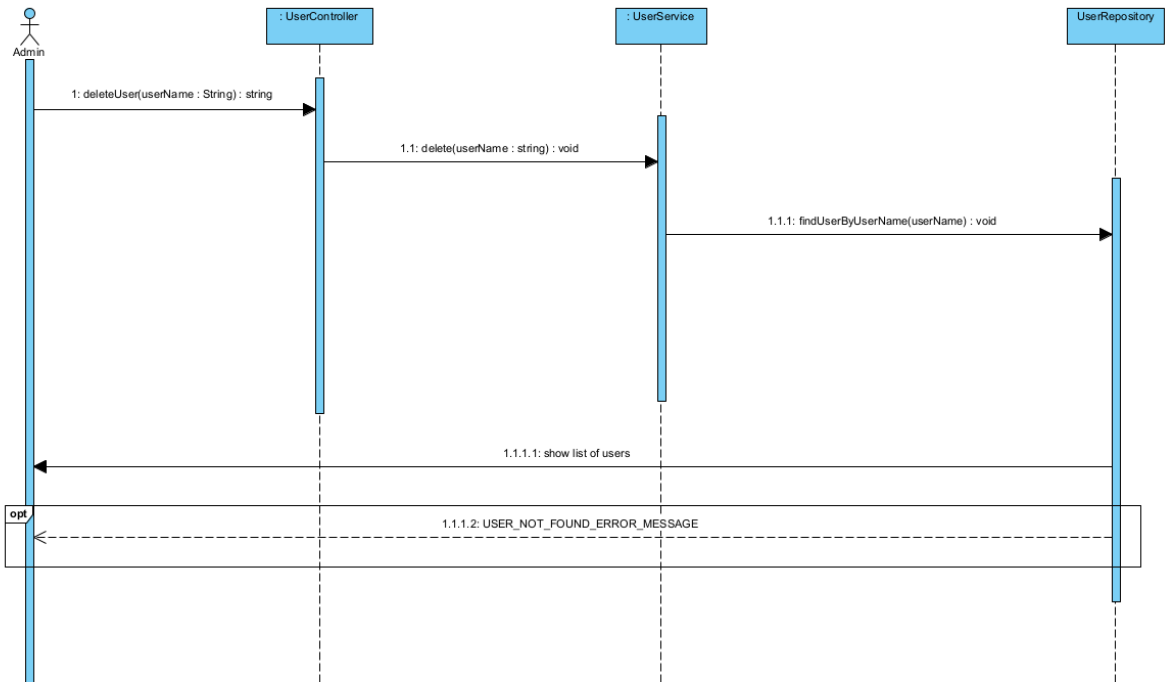
9.3. Wyszukiwanie samochodów



9.4 Usuwanie wiadomości



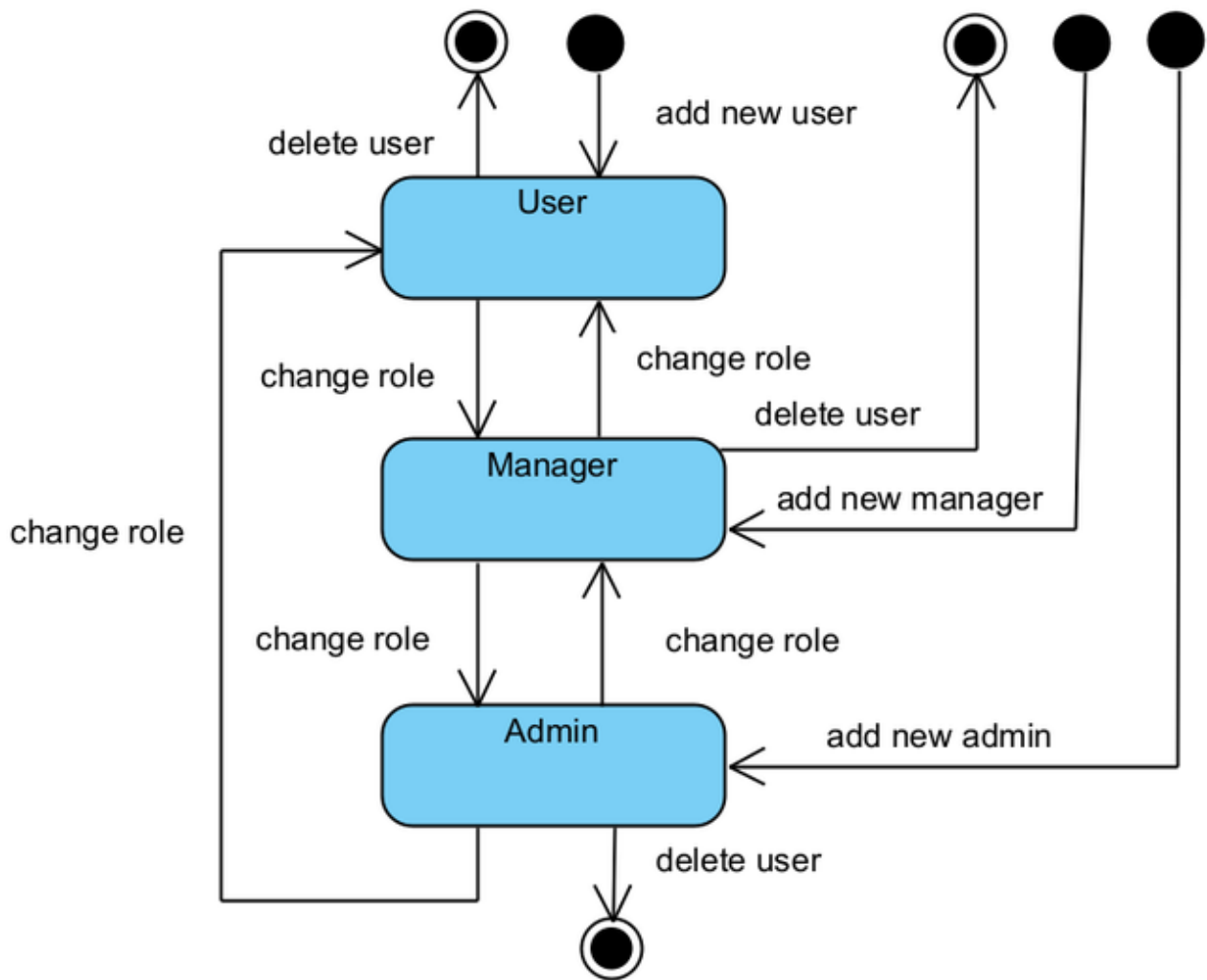
9.5 Usuwanie dowolnego użytkownika przez Administratora



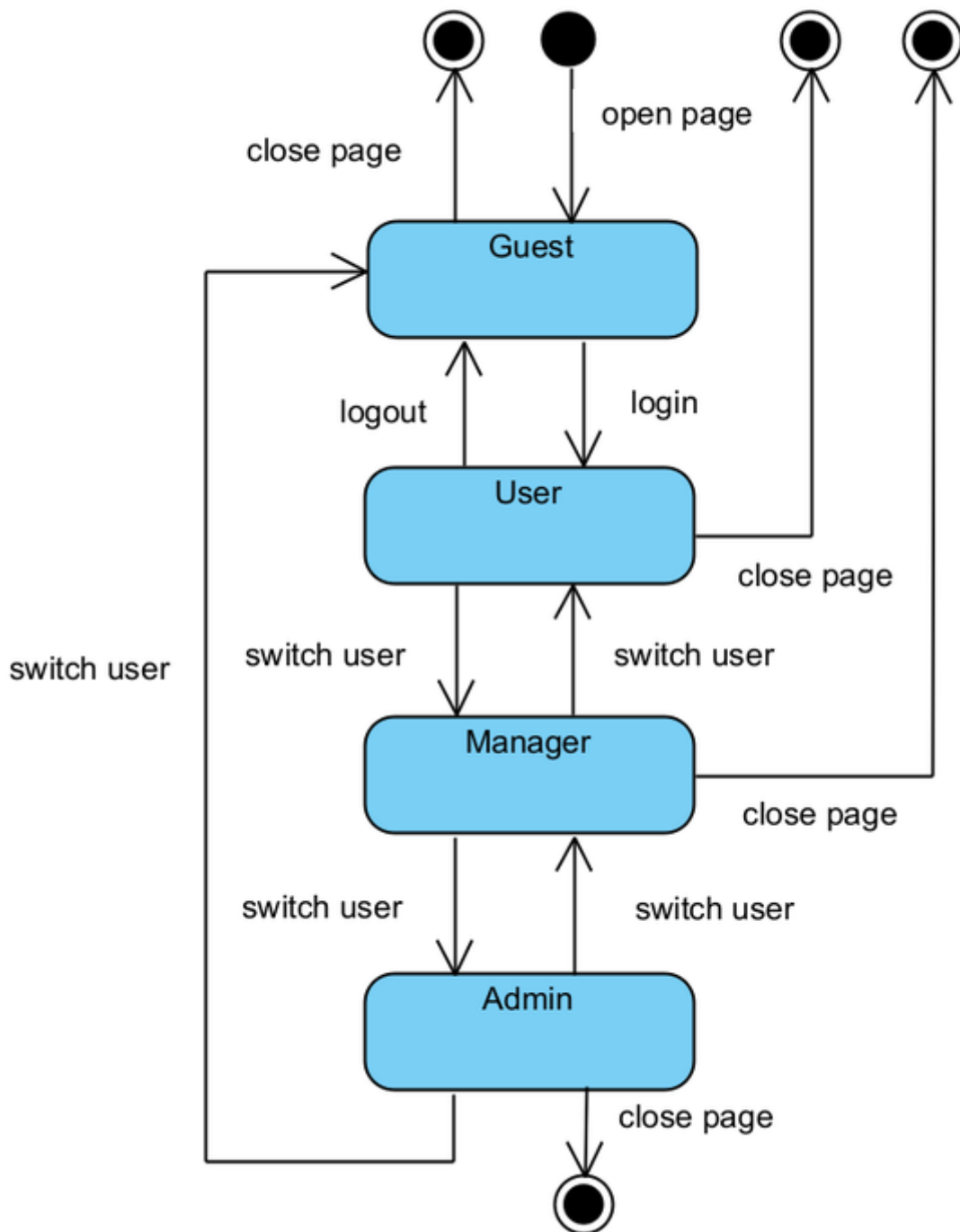
¹ <https://spring.io/projects/spring-boot>

10. Diagramy stanów (wybrane przykłady)

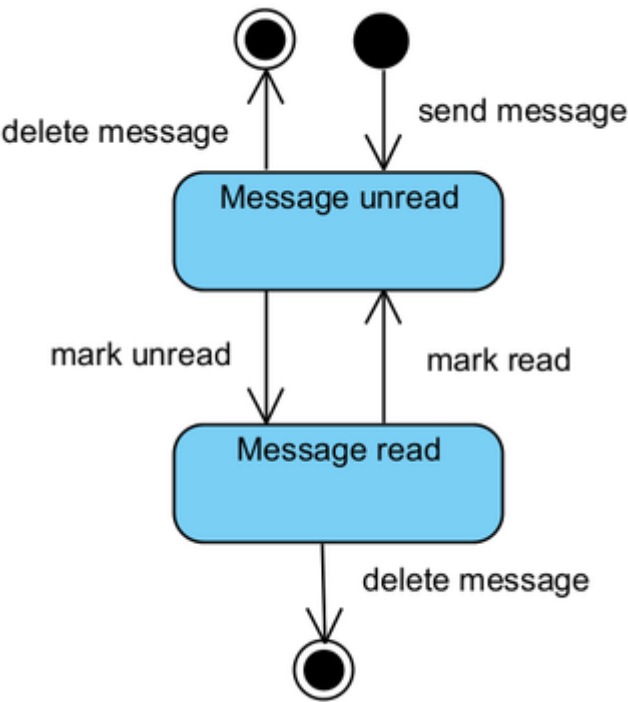
10.1. Zarządzanie użytkownikami (tylko Administrator)



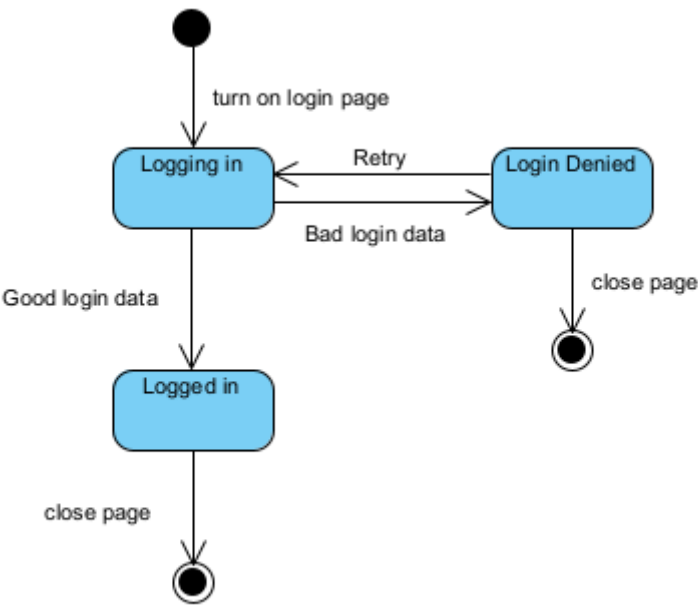
10.2. Status/rola zalogowanego użytkownika (tylko Administrator)



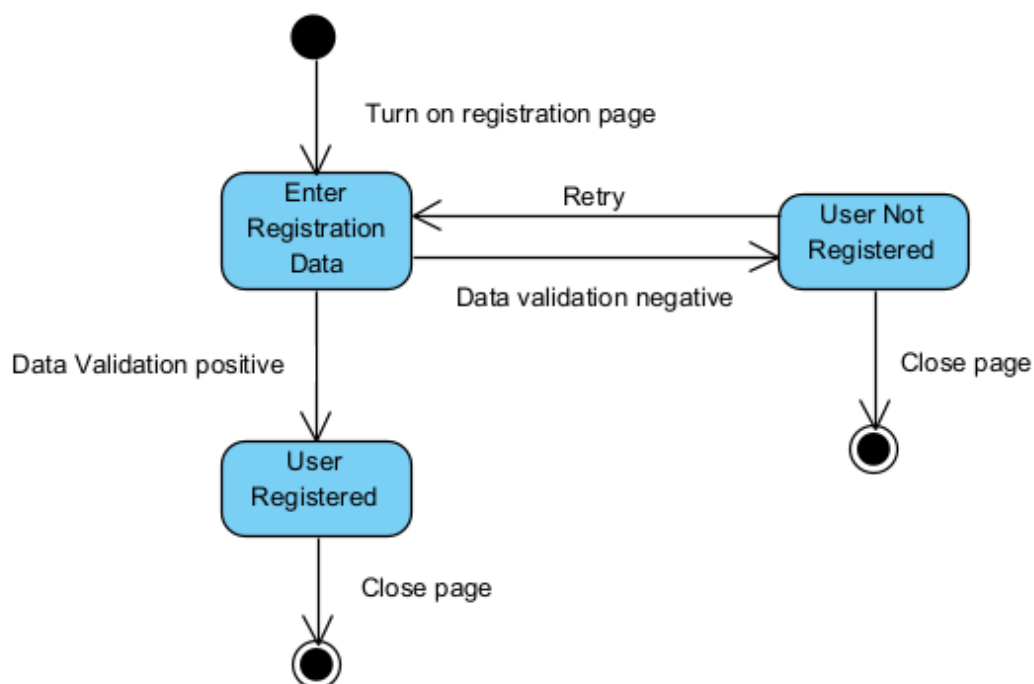
10.3. Zarządzanie wiadomościami (tylko *Zarządca/Administrator*)



10.4. Logowanie do systemu

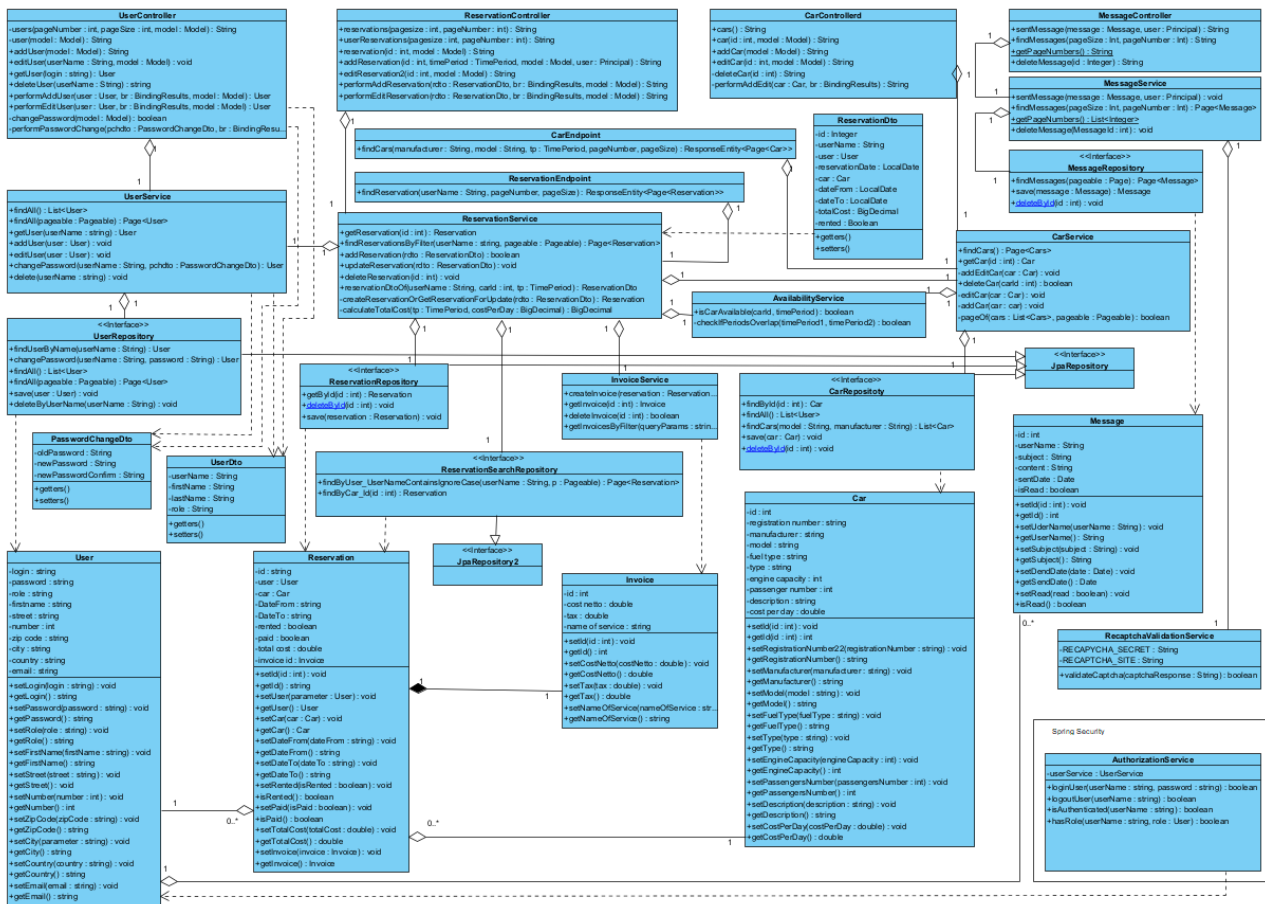


10.5. Rejestracja użytkownika



UWAGA Diagram rejestracji przedstawia sposób rejestracji dla zwykłego użytkownika. Wszelkie role specjalne (*Zarządca/Administrator*) są przypisywane ręcznie przez *Administratora*

11. Diagram klas



12. Kod SQL

12.1. Standard tworzenia bazy danych

Struktura bazy danych budowana jest z wykorzystaniem narzędzia Flyway (<https://flywaydb.org/>), które odpowiada również za wypełnienie bazy danych danymi testowymi. Z uwagi na fakt, że wersja MVP aplikacji nie posiada wszystkich projektowanych funkcji, również baza danych nie zawiera wszystkich tabel określonych na diagramie ERD. Poniżej zamieszczono kod SQL do wszystkich tabel, z których aktualnie korzysta aplikacja. Wszystkie tablice dostępne są również w folderze aplikacji: <https://github.com/lukaszse/car-rental/tree/master/src/main/resources/db/migration>

12.2. Dialekt SQL

W projekcie wykorzystano bazę danych H2 oraz dialekt SQL H2 (<https://www.h2database.com/>).

12.3. Kod SQL

12.3.1. Tabela CAR

```
create table car (  
    id int primary key auto_increment,  
    registration_no varchar(255),  
    manufacturer varchar(255) not null,  
    model varchar(255) not null ,  
    fuel_type varchar(255) not null ,  
    type varchar(255) not null ,  
    engine_capacity int not null ,  
    passenger_number int not null ,  
    description varchar(255) not null,  
    cost_per_day numeric(19,2) not null  
);
```

12.3.2. Tabela APP_USER

```
create table app_user (  
    user_name varchar(40) primary key,  
    first_name varchar(255),  
    last_name varchar(255),  
    user_role varchar(255),  
    password varchar(500)  
);
```

12.3.3. Tabela RESERVATION

```
create table reservation (  
    id int primary key auto_increment,  
    user_name varchar(255) not null,  
    car_id int not null,  
    reservation_date date,  
    date_from date,  
    date_to date,  
    total_cost numeric(19,2) not null,  
    invoice_id varchar(255),  
    rented bit,  
    foreign key (car_id) references car(id),  
    foreign key (user_name) references app_user(user_name)  
);
```

12.3.4. Tabela MESSAGE

```
create table message (  
    id int primary key auto_increment,  
    user_name varchar(40),
```

```
subject varchar(255),  
content varchar(1000),  
sent_date date,  
is_read bit  
);
```

12.3.5. Dane testowe

Aplikacja w wersji MVP po zainstalowaniu posiada w bazie danej przykładowe dane testowe pozwalające na sprawdzenie działania aplikacji. Dane te można usunąć, po przekazaniu aplikacji do użytkowania. W celu zapisania danych w bazie wykorzystano mechanizm migracji Flyway. Kod służący do dodania danych testowych (wraz z kodem do stworzenia tabel) znajduje się w poniższej lokalizacji: <https://github.com/lukaszse/car-rental/tree/master/src/main/resources/db/migration>

13. Informacje dla developera

13.1. Zastosowane technologie i wymagania wobec developera

Aplikację napisano w języku Java w wersji 17 oraz z wykorzystaniem frameworku Spring Boot. W aplikacji wykorzystano także mechanizm szablonów Thymeleaf oraz elementy napisane w języku JavaScript. Podstawowa znajomość wszystkich tych technologii jest konieczna do rozpoczęcia pracy z kodem aplikacji. Przed rozpoczęciem pracy z kodem należy:

1. Zainstalować środowisko OpenJDK 17.
2. Zainstalować wybrane środowisko IDE (np. IntelliJ lub Eclipse).
3. Zainstalować aplikację Docker.

13.2. Korzystanie z repozytorium

Aby skorzystać z repozytorium, należy na lokalnej maszynie zainstalować aplikację Git do kontroli wersji. Aplikacja jest dostępna dla systemów Windows, Linux oraz MacOS: <https://git-scm.com/> Kod źródłowy aplikacji Car-Rental znajduje się w repozytorium w serwisie GitHub: <https://github.com/lukaszse/car-rental>

W celu pobrania repozytorium użyj komendy:

```
git clone https://github.com/lukaszse/car-rental.git
```

13.3. Uruchamianie aplikacji

Aby uruchomić aplikację, należy zbudować plik jar. W tym celu należy użyć komendy:

```
./mvnw clean install
```

plik jar zostanie stworzony w folderze `/target`. Plik jar można uruchomić przy pomocy komendy:

```
java -jar nazwa_pliku.jar
```

13.4. Konfiguracja

W aplikacji skonfigurowano dwa profile **LOCAL** służący do uruchamiania aplikacji na lokalnym komputerze (z rozszerzonymi opcjami logowania, debugowania oraz dostępem bez szyfrowania TLS) oraz **PROD** służący do uruchomienia aplikacji produkcyjnej na serwerze. Profil można przełączyć poprzez modyfikację zmiennej

`spring.profiles.active=prod`, która znajduje się w pliku

`src/main/resources/application.properties`. Szczegółowe konfiguracje dla środowisk znajdują się w plikach `application-local.yml` oraz `application-prod.yml` znajdujących się w tej samej lokalizacji.

13.5 Obraz Docker

W głównym folderze aplikacji znajduje się plik `Dockerfile`, który służy do zbudowania obrazu z aktualnego pliku jar znajdującego się w folderze `target`. Aby zbudować obraz name użyć komendy:

```
docker build -t nazwaObrazu .
```

W celu zapisania obrazu w repozytorium zdalnym Docker'a (tak, aby był dostępny on online dla publicznie), należy użyć komendy:

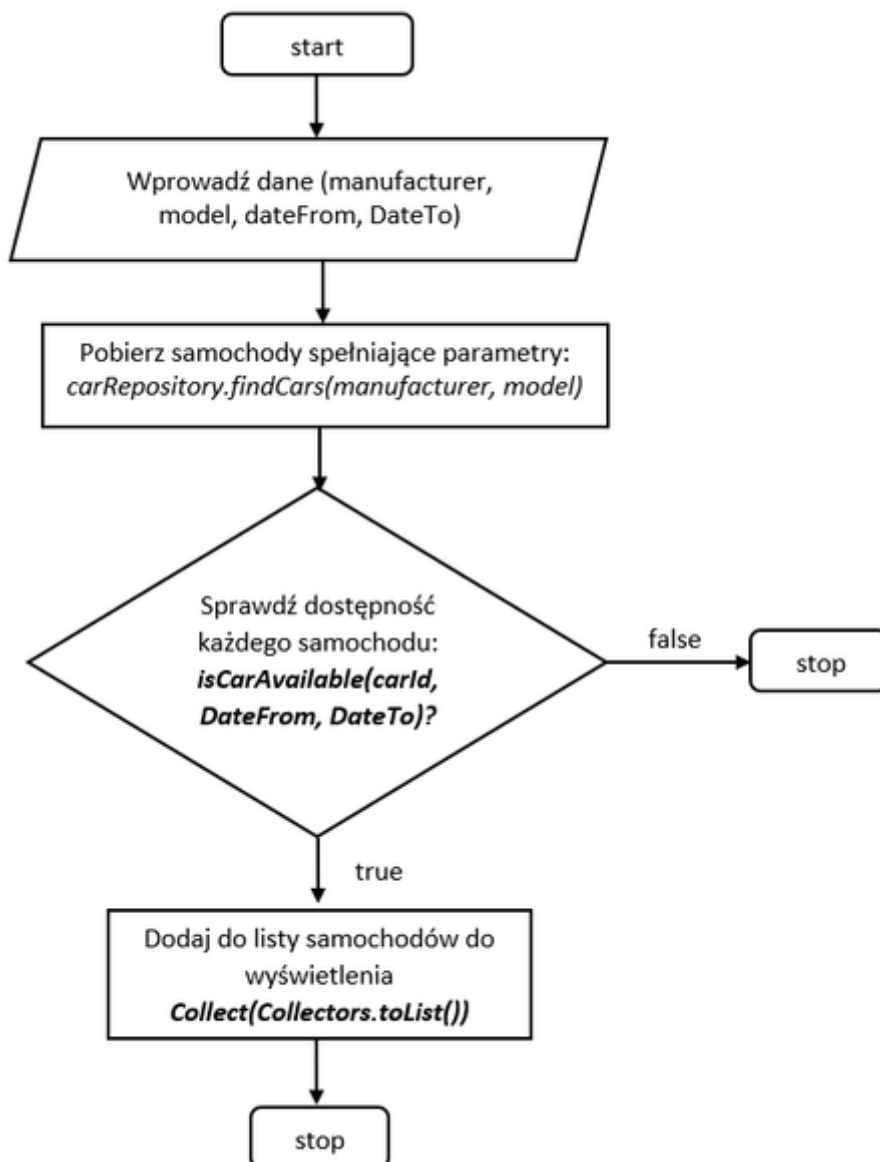
```
docker push nazwaObrazu
```

13.6. Wybrane szczegóły implementacji - algorytm sprawdzania dostępności samochodu w danym przedziale czasu

W aplikacji zastosowano mechanizm sprawdzania dostępności samochodu w danym przedziale czasu. Algorytm ten wykorzystywany jest w dwóch sytuacjach:

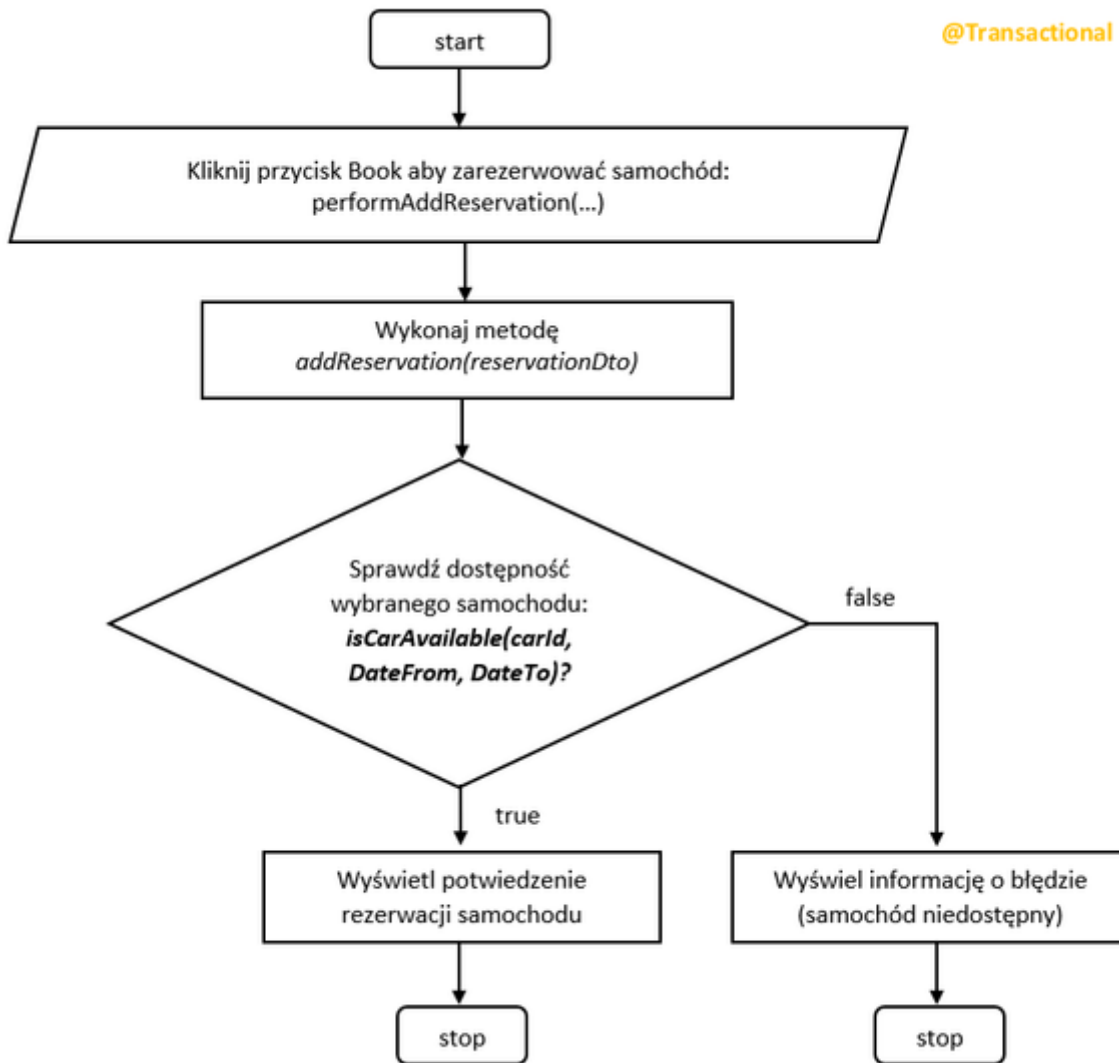
1. Wyszukiwanie samochodów w widoku **cars** - użytkownik ma możliwość wpisania zakresu czasu (`dateFrom` oraz `dateTo`), w celu wyszukiwania dostępnych do wypożyczenia samochodów. Mechanizm ten realizowany jest z wykorzystaniem zapytania REST API **@GET**, które zaimplementowane zostało z wykorzystaniem

Javascript (AJAX). Do obsługi zapytania @GET utworzono endpoint `cars/findCars`, który przyjmuje m.in. dwa parametry QueryParam - `dateFrom` oraz `dateTo`.



3. Rezerwowanie samochodu (dodawanie rezerwacji) - po wyszukaniu samochodu użytkownik w widoku `cars` ma możliwość zarezerwowania wybranego samochodu przez kliknięcie przycisku **Book**, a następnie poprzez potwierdzenie zamówienia przez kliknięcie przycisku **Submit** na ekranie z danymi samochodu, co spowoduje wysłanie zapytania @POST oraz wywołanie metody 'performAddReservation' która przyjmuje kilka paramerów, w tym parametry `dateFrom` oraz `dateTo`. Przed zarezerwowaniem samochodu również sprawdzana jest dostępność samochodu, a operacja przeprowadzana jest w formie transakcji.

@Transactional



Obie wyżej wymienione metody korzystają z klasy `AvailabilityService` oraz zaimplementowanego w niej algorytmu:

```

public boolean isCarAvailable(final Integer carId, final TimePeriod timePeriod) {
    if(timePeriod.getDateFrom() == null && timePeriod.getDateTo() == null) {
        return true;
    }
    return reservationSearchRepository.findByCar_Id(carId).stream()
        .map(reservation -> TimePeriod.of(reservation.getDateFrom(), reservat
        .noneMatch(reservationPeriod -> checkIfPeriodOverlap(timePeriod, rese
}

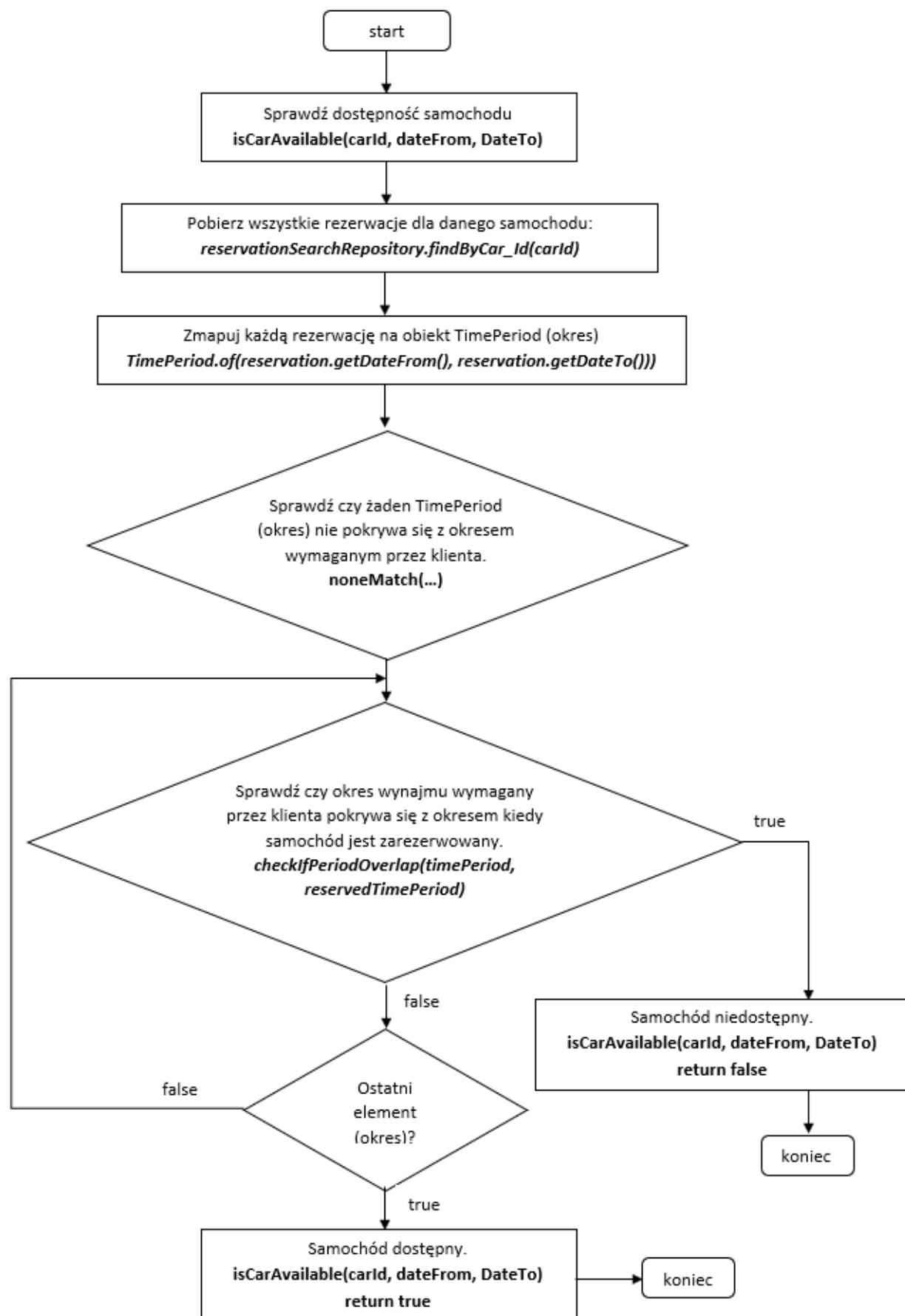
private static boolean checkIfPeriodOverlap(final TimePeriod timePeriod1, TimePer
    final LocalDate s1 = timePeriod1.getDateFrom();
    final LocalDate e1 = timePeriod1.getDateTo();
    final LocalDate s2 = timePeriod2.getDateFrom();
    final LocalDate e2 = timePeriod2.getDateTo();

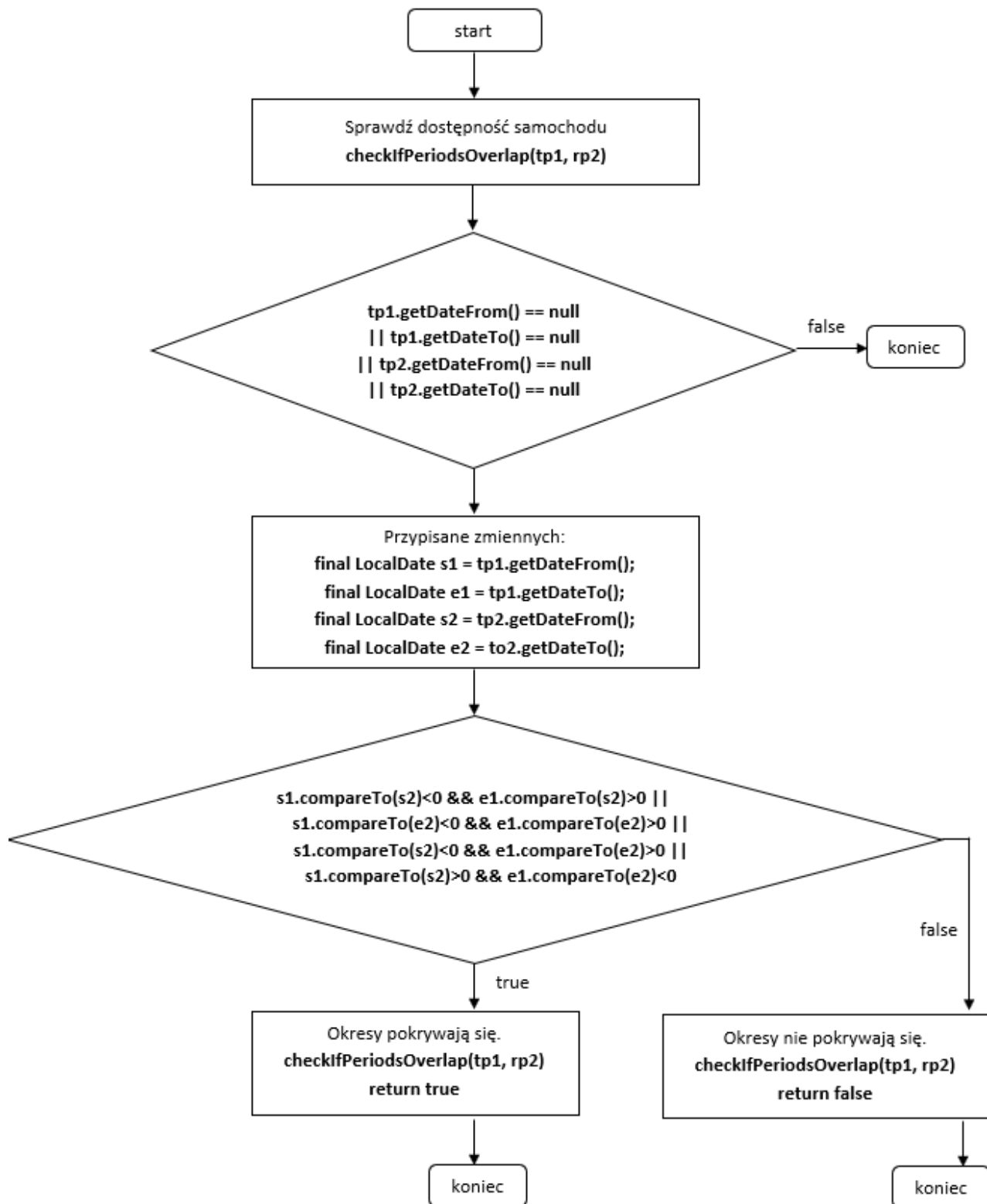
    if(s1 == null || e1 == null || s2 == null || e2 == null) {
        return false;
    }
}

```

```
        if(s1.compareTo(s2)<=0 && e1.compareTo(s2)>=0 ||
           s1.compareTo(e2)<0 && e1.compareTo(e2)>0 ||
           s1.compareTo(s2)<0 && e1.compareTo(e2)>0 ||
           s1.compareTo(s2)>0 && e1.compareTo(e2)<0 )
        {
            log.info("Periods overlap! Period 1: {}, Period2: {}", timePeriod1, timeF
            return true;
        }
        else {
            return false;
        }
    }
}
```

Jak widać (co wynika z powyższego kodu) metoda `isCarAvailable` wywołuje zapytanie bazy danych z wykorzystaniem `RepositorySearchService` w celu pobrania wszystkich rezerwacji dla danego samochodu, a następnie sprawdza, czy jakikolwiek okres z pobranych rezerwacji nie pokrywa się z okresem rezerwacji wymaganym przez użytkownika. Jeśli żaden z tych okresów się nie pokrywa `noneMatch` metoda zwraca `true`, w przeciwnym razie `false`. Samo sprawdzenie, czy pojedynczy pobrany z bazy danych okres rezerwacji pokrywa się z okresem rezerwacji wymaganym przez użytkownika, sprawdzane jest w metodzie `'checkIfPeriodOverlap'`. Niniejszy algorytm przedstawiono na poniższych schematach blokowych (zastosowano osobny schemat dla metody `checkIfPeriodsOverlap`):





14. Testowanie

Podstawową formą testów aplikacji będą testy jednostkowe oraz integracyjne pisane na bieżąco, w trakcie powstawania kodu źródłowego, pisane przez zespół testerski. Co więcej, w uzgodnieniu z użytkownikami aplikacji przygotowane zostanie około 20 różnych przypadków testowych. Testy realizować będzie zespół testerów. W tym celu utworzone zostanie środowisko testowe, na którym zostanie uruchomiona pełna funkcjonalność aplikacji. Przypadki testowe będą zawierały m.in. przypadki dodawania nowych samochodów, dodawania użytkowników, rezerwacji samochodów itd. Dodatkowo zostaną przeprowadzone testy wydajnościowe aplikacji dla określonej grupy wirtualnych użytkowników.

14.1. Testy jednostkowe

W aplikacji wykorzystano testowy framework Spock oraz testy jednostkowe napisane w języku Groovy. Spock umożliwia między innymi tworzenie testów wykorzystujących koncepcję Data Driven Tests.

14.1.1 Testowanie metody sprawdzającej dostępność samochodów

Jak opisano w punkcie 14.2.1., w aplikacji zastosowano algorytm sprawdzania dostępności pojazdów, który znajduje się w klasie `AvailabilityService`. Algorytm ten wykorzystuje metodę `checkIfPeriodsOverlap`, która sprawdza, czy dwa okresy się pokrywają (okres 'TimePeriod' jest obiektem zawierającym dwie daty

- datę "od" oraz datę "do"). Poniżej zamieszczono kod testu jednostkowego sprawdzającego poprawność działania metody `checkIfPeriodsOverlap`:

```
@Unroll
def "should check if data ranges overlap correctly - test #no"() {

    when: "invoke method to check if data ranges overlaps"
    def overlap = availabilityService.checkIfPeriodsOverlap(firstPeriod, secondPe

    then: "should check correctly if data ranges overlap"
    overlap == expectedResult

    where:
    no | firstPeriod          | secondPeriod          || expectedResult
    1 | getTimePeriod(1, 2)   | getTimePeriod(3, 4)   || false
    2 | getTimePeriod(2, 4)   | getTimePeriod(3, 5)   || true
    3 | getTimePeriod(1, 4)   | getTimePeriod(20, 30) || false
    4 | getTimePeriod(10, 14) | getTimePeriod(7, 13)  || true
    5 | getTimePeriod(10, 14) | getTimePeriod(7, 60)  || true
    7 | getTimePeriod(1, 200) | getTimePeriod(7, 60)  || true
    8 | getTimePeriod(1, 3)   | getTimePeriod(1, 3)   || true
    9 | getTimePeriod(5, 8)   | getTimePeriod(1, 3)   || false
}
```

```
static def getTimePeriod(final int plusDaysFrom, final int plusDaysTo) {
    TimePeriod.of(LocalDate.now().plusDays(plusDaysFrom), LocalDate.now().plusDay
}
```

Aby uruchomić test, konieczne było utworzenie protez ("mocks") dla wykorzystanych obiektów:

```
ReservationSearchRepository repository = Mock()
AvailabilityService availabilityService = new AvailabilityService(repository)
```

Odnosnik do klasy testowej: <https://github.com/lukaszse/car-rental/blob/master/src/test/groovy/org/lukaszse/carRental/service/AvailabilityServiceSpec.groovy>

14.1.2 Walidacja daty

W aplikacji wykorzystano mechanizm adnotacji do walidowania m.in. danych przychodzących z zewnątrz (z przeglądarki internetowej do serwera). Stworzono, także m.in. niestandardową adnotację `@ValidateTimePeriod` oraz walidator do sprawdzania poprawności wprowadzanych dat. Walidator stanowi odrębną klasę, z główną metodą `isValid`, która używa odpowiedniej logiki do zweryfikowania poprawności wprowadzonego okresu (`TimePeriod`). Walidator sprawdza m.in. czy nie wprowadzono daty z przeszłości oraz, czy data "od" nie jest wcześniejsza niż data "do". W celu sprawdzenia poprawności działania walidatora przygotowano test jednostkowy weryfikujący poprawność działania metody `isValid`:

```
def "should validate TimePeriod correctly"() {

    expect: 'should return correct validation result'
    timePeriodValidator.isValid(timeperiod, constraintValidatorContext) == expect

    where:
    timeperiod
    TimePeriod.of(LocalDate.now().plusDays(2), LocalDate.now().plusDays(3))
    TimePeriod.of(LocalDate.now().plusDays(5), LocalDate.now().plusDays(3))
    TimePeriod.of(LocalDate.now().minusDays(5), LocalDate.now().plusDays(3))
    TimePeriod.of(LocalDate.now().minusDays(5), LocalDate.now().minusDays(3))
    TimePeriod.of(LocalDate.now().plusDays(3), LocalDate.now().plusDays(3))
}
```

Metoda testowa wykorzystuje protezy obiektów ("mocks"):

```
TimePeriodValidator timePeriodValidator = new TimePeriodValidator();
ConstraintValidatorContext constraintValidatorContext = Mock()
```


14.1.3 Testowanie metody obliczającej koszt całkowity rezerwacji

W aplikacji wykorzystano metodę statyczną `calculateTotalCost` służącą do obliczania kosztu całkowitego rezerwacji w danym przedziale czasowym, który został wybrany wcześniej przez użytkownika, i po cenie rezerwacji za dzień, określonej przez administrację serwisu. W celu sprawdzenia poprawności działania metody przygotowano test jednostkowy:

```
def "should check if data ranges equals correctly - test #no"() {

    given: "Prepare dates"
    def localDateFrom = LocalDate.parse(dateFrom)
    def localDateTo = LocalDate.parse(dateTo)

    when: "Try to calculate cost"
    def overlap = ReservationService.calculateTotalCost(localDateFrom, localDateTo)

    then: "Cost should be calculated correctly"
    overlap == expectedResult

    where:
    no | dateFrom      | dateTo      | costPerDay | || expectedResult
    1 | "2018-05-20" | "2018-06-02" | 100        | || 1300
    2 | "2019-03-10" | "2019-05-22" | 70         | || 5110
    3 | "2018-03-20" | "2018-03-22" | 200        | || 400
    4 | "2000-09-01" | "2000-09-30" | 50         | || 1450
}
```

14.1.4 Testowanie metody zmieniającej hasło użytkownika

W klasie `UserService` znajduje się metoda `changePassword`, jest to metoda pomocnicza służąca do zmiany hasła użytkownika. Poniżej zamieszczono kod testu jednostkowego sprawdzającego poprawność działania metody:

```
def "should change password"() {

    given: "Mock all required methods"
    userRepository.findUserByUsername(_ as String) >> Optional.of(prepareUser())
    passwordEncoder.matches(_ as String, _ as String) >> true
    passwordEncoder.encode(_ as String) >> "some encrypted password"
    userService.passwordEncoder = passwordEncoder

    and: "prepare prepareChangePasswordDto object"
    def passwordChangeDto = prepareChangePasswordDto(oldPassword, newPassword, ne
```

```

when: "try to change password"
userService.changePassword(userName, passwordChangeDto)

then: "should throw now exception and invoke changePassword() method once"
noExceptionThrown()
1 * userRepository.changePassword(_ as String, _ as String)

where:
no | userName | oldPassword | newPassword | newPasswordConfirm
1 | "joe"     | "password"  | "qwerty"   | "qwerty"
2 | "joe"     | "password"  | "123456"   | "123456"
}

def "should throw exception while trying to change password with wrong data"() {

    given: "Mock all required methods"
    userRepository.findUserByUserName(_ as String) >> Optional.of(prepareUser())
    passwordEncoder.matches(_ as String, _ as String) >> passwordEncoderMatch
    passwordEncoder.encode(_ as String) >> "some encrypted password"
    userService.passwordEncoder = passwordEncoder

    and: "prepare prepareChangePasswordDto object"
    def passwordChangeDto = prepareChangePasswordDto(oldPassword, newPassword, ne

    when: "try to change password"
    userService.changePassword(userName, passwordChangeDto)

    then: "should throw exception and not change password"
    thrown(WrongPayloadException.class)
    0 * userRepository.changePassword(_ as String, _ as String)

    where:
    no | userName | oldPassword | newPassword | newPasswordConfirm | passwordEncc
    1 | "joe"     | "password"  | "qwerty2"   | "qwerty"           | true
    2 | "joe"     | "password"  | "1234562"   | "123456"           | true
    3 | "joe"     | "password"  | "123456"    | "123456"           | false
}

def static prepareChangePasswordDto(String oldPassword, String newPassword, Strir
def passwordChangeDto = new PasswordChangeDto()
passwordChangeDto.setOldPassword(oldPassword)
passwordChangeDto.setNewPassword(newPassword)
passwordChangeDto.setNewPasswordConfirm(newPasswordConfirm)
passwordChangeDto
}

def static prepareUser() {
    def user = new User()
    user.setUserName("Joe")
    user.setPassword('{bcrypt}$2a$12$feoSS.Dx/rRdQWfWHewYZu8txsYcy8Dxt89Mwd9U3C
    // Password = password
    user

```

```
}  
}
```

Aby uruchomić test, konieczne było utworzenie protez ("mocks") dla wykorzystanych obiektów:

```
PasswordEncoder passwordEncoder = Mock()  
UserRepository userRepository = Mock()  
UserService userService = new UserService(userRepository)
```

14.2 Przypadki testowe dla testów manualnych

14.2.1. Logowanie do aplikacji

Cel: Sprawdzenie możliwości zalogowania dla użytkowników o różnych uprawnieniach

Warunki początkowe

- W aplikacji istnieją aktywne konta użytkowników dla każdej roli tj. użytkownika, *Administradora* i *Zarządcę*.
- Użytkownik znajduje się na ekranie powitalnym aplikacji

Krok	Rezultat
1. Kliknij na Sign In (menu)	1'. Wyświetlono ekran logowania
2. Wprowadź login i hasło	2'. Uzupełniono pola formularza
3. Naciśnij przycisk zaloguj	3'. Zalogowano do systemu

Priorytet: wysoki

Wykonanie manualne

Szacowany czas: 1 min. dla każdego logowania

Uwagi: powtórzyć dla każdej roli [user, manager, admin]

14.2.2. Wyszukiwanie samochodu

Cel: Sprawdzenie możliwości wyszukania samochodu

Warunki początkowe

- Użytkownik zalogowany w systemie [user, manager, admin]

- W bazie danych znajduje się dostępne samochody (utworzone automatycznie dla celów testowych)
- Przed rozpoczęciem testu sprawdzić w bazie danych samochody dostępne dla wybranego okresu

Krok	Rezultat
1. Kliknij na <code>Cars</code> (menu)	1'. Wyświetlono z dostępnymi samochodami
2. Wprowadź wybrany okres (<code>dateFrom</code> i <code>dateTo</code>)	2'. Wyświetlono samochody dostępne w danym okresie
3. Wprowadź pierwsze znaki marki samochodu i naciśnij <code>enter</code>	3'. Wyświetlono samochody których marka rozpoczyna się od wprowadzonych liter
4. Wprowadź pierwsze znaki modelu samochodu i naciśnij <code>enter</code>	4'. Wyświetlono samochody których model rozpoczyna się od wprowadzonych liter
5. Kliknij przycisk <code>View Details</code> dla wybranego samochodu	5'. Wyświetlono szczegóły samochodu

Priorytet: wysoki

Wykonanie manualne

Szacowany czas: 1 min. dla każdej roli + 2 min. na sprawdzenie danych w bazie.

Uwagi: powtórzyć dla każdej roli [user, manager, admin]

14.2.3. Rejestracja

Cel: Sprawdzenie możliwości rejestracji *Gościa*

Warunki początkowe

- Wejście na stronę jako użytkownik niezarejestrowany w bazie.
- Użytkownik znajduje się na ekranie powitalnym aplikacji

Krok	Rezultat
1. Kliknij na <code>Sign Up</code> (menu)	1'. Wyświetlono ekran rejestracji
2. Wprowadź login i hasło	2'. Uzupełniono formularz rejestracji
3. Naciśnij <code>Submit</code>	3'. Następuje rejestracja

Priorytet: wysoki

Wykonanie manualne

Szacowany czas: 1 min. **Uwagi:** Brak

14.2.4. Wysyłanie wiadomości przez niezalogowanego użytkownika

Cel: Sprawdzenie możliwości wysłania wiadomości

Warunki początkowe

- Użytkownik niezalogowany w systemie
- Użytkownik znajduje się na ekranie powitalnym aplikacji

Krok	Rezultat
1. Kliknij na <code>Send Message</code> (menu)	1. Wyświetlono ekran wysyłania wiadomości
2. Wprowadź imię, temat i treść, zatwierdź captcha	2. Uzupełniono formularz wysyłania wiadomości
3. Naciśnij <code>Submit</code>	3. Następuje wysłanie wiadomości

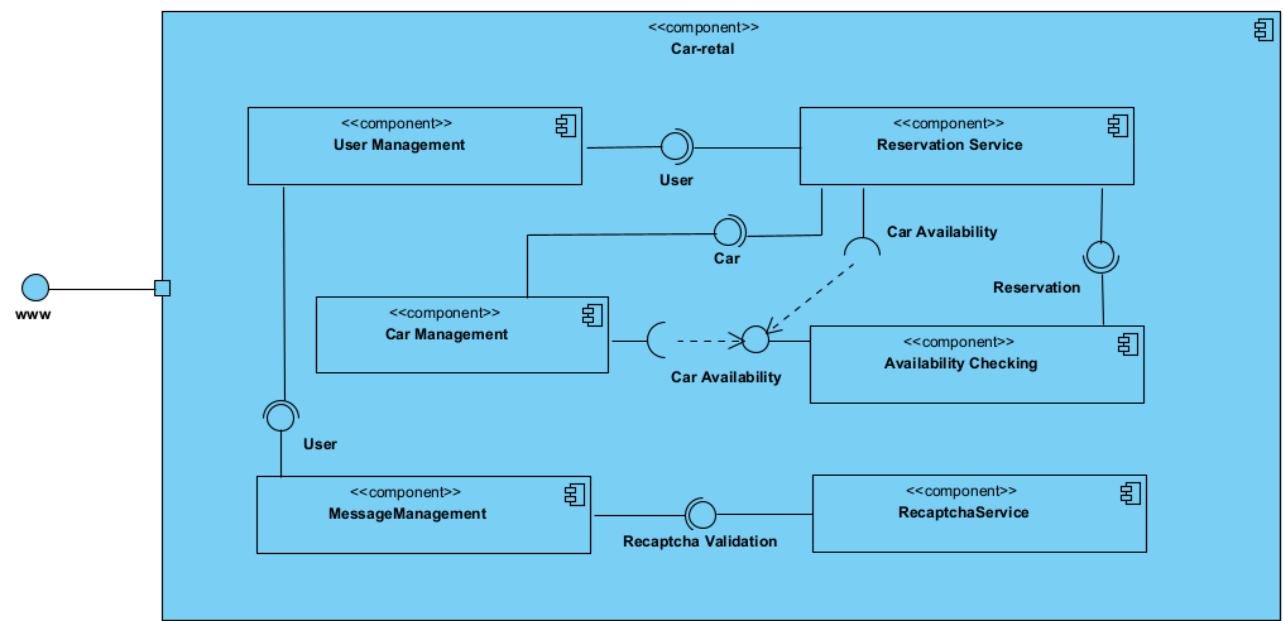
Priorytet: niski

Wykonanie manualne

Szacowany czas: 1 min

Uwagi: Brak

15. Diagram głównych komponentów systemu

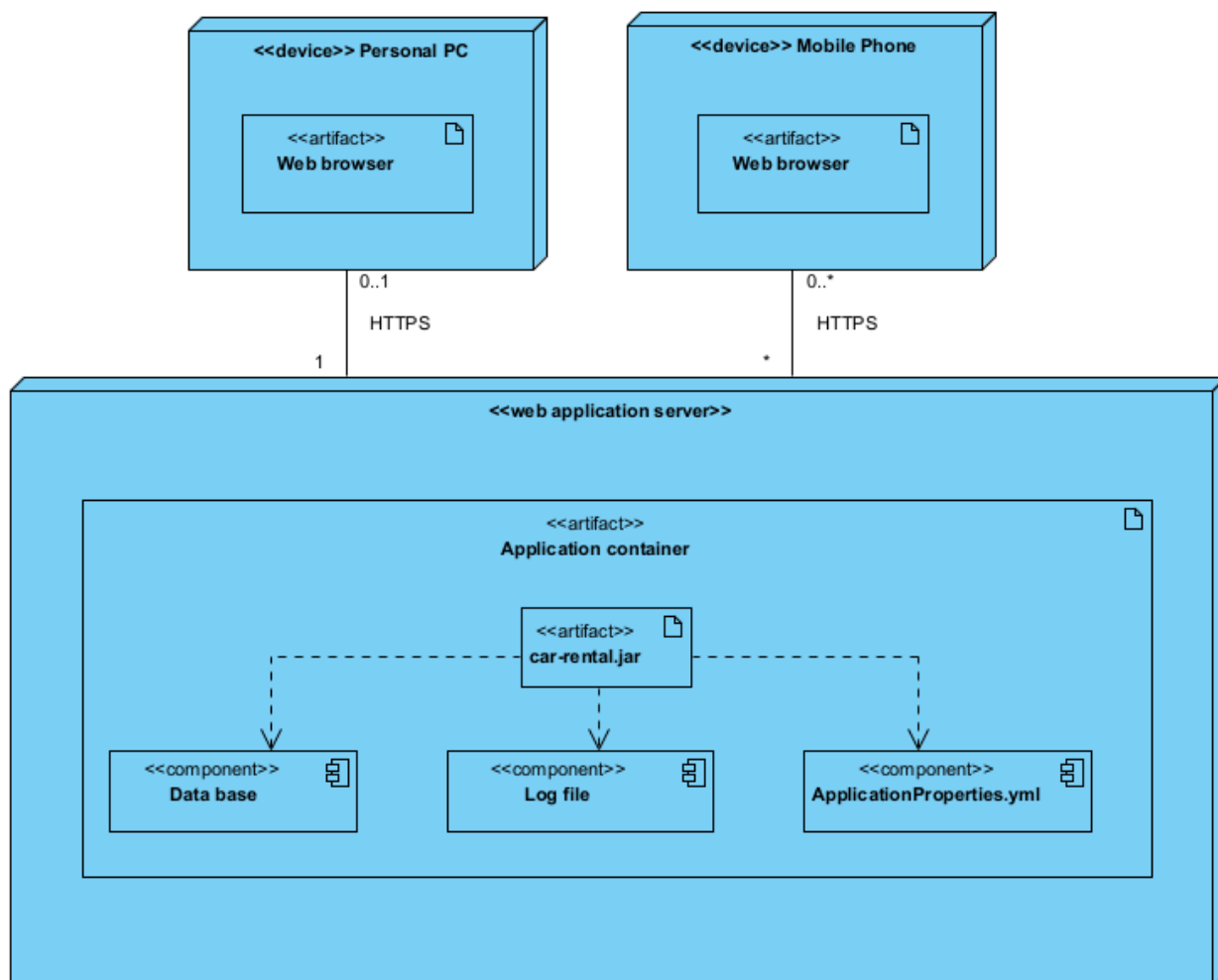


Powyższy diagram komponentów przedstawia główne komponenty systemu z wyłączeniem bazy danych.

16. Wdrożenie

16.1. Diagramy wdrożenia

16.1.1. Wdrożenie z wykorzystaniem kontenera Docker



Uwaga W wersji MVP aplikacja może wyświetlać się niepoprawnie na urządzeniach mobilnych, ze względu na brak skalowania tabel

16.2. Wymagania systemowe

Aplikacja napisana została w wieloplatformowym języku Java. Działa na każdym systemie z systemem operacyjnym Windows, Linux czy MacOS. Poniżej przedstawiono szczegółowe wymagania systemowe.

Wymagania systemowe:

- System operacyjny Windows 10/11, MacOS, Linux oraz inne systemy z rodziny Unix.
- Zainstalowana maszyna wirtualna Javy w wersji minimum 17 (JRE / JDK 17) w przypadku uruchamiania aplikacji z pliku jar.
- Zainstalowane oprogramowanie Docker w przypadku uruchamiania aplikacji z obrazu docker. Zalecana wersja 20.10.10 lub wyższa.

16.3. Instalacja z wykorzystaniem pliku jar

Skopiować plik na serwer oraz uruchomić komendę:

```
java -jar nazwa_pliku.jar
```

16.4. Instalacja z wykorzystaniem obrazu Docker

Aby ściągnąć obraz Dockera zawierający aplikację, należy użyć kolejno komend:

```
docker pull llseremak/car-rental
```

Aby pobrać obraz, a następnie:

```
docker run -d --restart unless-stopped -p 443:443 llseremak/car-rental
```

gdzie pierwszy port 443 to port, pod którym aplikacja będzie dostępna z zewnątrz kontenera (port 443 jest portem domyślnym dla połączeń szyfrowanych z wykorzystaniem TLS)

16.5. Dodatkowa konfiguracja z wykorzystaniem NGINX

Przy pomocy NGINX można skonfigurować przekierowanie z portu, na którym działa aplikacja do określonego adresu url. Dokumentacja NGINX: <http://nginx.org/en/docs/>

16.5. Bezpieczeństwo i certyfikat HTTPS

W wersji demonstracyjnej aplikacji wykorzystano niezarejestrowany certyfikat HTTPS. Gwarantuje on szyfrowanie danych przesyłanych z przeglądarki do serwera, jednak nie jest to certyfikat wydany przez Urząd Certyfikacji, wobec czego nie będzie traktowany przez przeglądarkę jako certyfikat zaufany.

Wymagać to może, odpowiednich kroków w zależności od konkretnej przeglądarki. W większości przypadków konieczne będzie wybranie opcji zaawansowanych[1], w celu wyświetlenia możliwości otwarcia strony[2].



Twoje połączenie nie jest prywatne

Atakujący mogą próbować ukraść Twoje informacje z witryny **ubuntu.llseremak.p3.tiktalik.io** (na przykład hasła, wiadomości lub karty kredytowe).

NET::ERR_CERT_AUTHORITY_INVALID

Ukryj zaawansowane

1

Wróć

Ten serwer nie może udowodnić, że jest to **ubuntu.llseremak.p3.tiktalik.io**; jego certyfikat zabezpieczeń nie jest zaufany przez system operacyjny komputera. Może to być spowodowane błędną konfiguracją lub przechwyceniem połączenia przez atakującego.

[Przejdź do witryny ubuntu.llseremak.p3.tiktalik.io \(niebezpieczna\)](#)

2

17. Podręcznik użytkownika

Spis treści

1. Rejestrowanie użytkownika
2. Logowanie do systemu
3. Wyszukiwanie dostępnych pojazdów
4. Składanie rezerwacji
5. Przeglądanie rezerwacji oraz usuwanie rezerwacji
6. Wysyłanie wiadomości
7. Funkcje dostępne dla *Zarządcy*
 - i. Edycja pojazdów
 - ii. Usuwanie pojazdów
 - iii. Przeglądanie rezerwacji wszystkich użytkowników
 - iv. Edycja rezerwacji
 - v. Usuwanie rezerwacji
 - vi. Odczytywanie wiadomości
8. Funkcje dostępne dla *Administradora*
 - i. Zarządzanie użytkownikami
 - a. Dodawanie użytkownika

- b. Edycja użytkownika
- ii. Ustawienia administracyjne

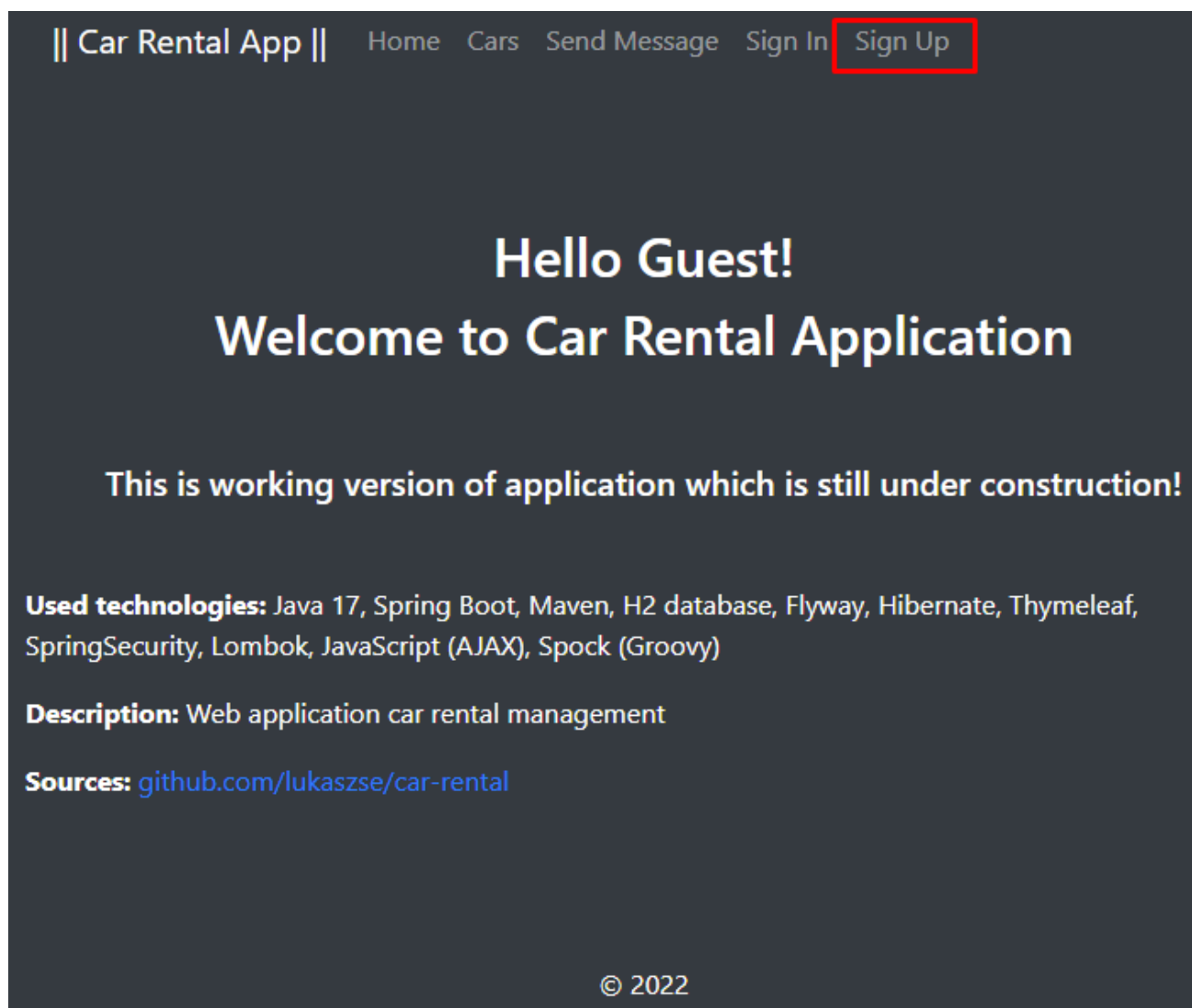
Wstęp

Aby zapewnić odpowiednią czytelność strony, autorzy aplikacji skorzystali ze standardowej biblioteki stylów Bootstrap (<https://getbootstrap.com/>)

17.1. Rejestrowanie użytkownika

W celu rejestracji należy wejść na stronę aplikacji: <https://ubuntu.lseremak.p3.tiktalik.io/car-rental>

Po przekierowaniu do strony startowej należy kliknąć zakładkę `Sign Up` w głównym menu w celu przekierowania do formularza rejestracyjnego.



Następnie należy uzupełnić formularz rejestracyjny.

Add User:

Login

First Name

Last Name

Password

Submit

Reset

Wpisane przez użytkownika dane są walidowane. W przypadku wpisania błędnych lub niepełnych danych zostanie zwrócony komunikat o błędzie.

Add User:

Error. Wrong data format:
Password cannot be blank
Password must contain at least 6 characters
Login must start with a letter and contain 2 - 20 word characters (digits, letters, _)
First name cannot be blank
Login cannot be blank
Last name cannot be blank

Login

First Name

Last Name

Password

Submit

Reset

Komunikat wystąpi także, jeśli użytkownik o danym loginie już istnieje.

Error. Wrong data format:
User with name user already exist

Login

user

First Name

John

Last Name

Doe

Password

Submit

Reset

Po wpisaniu prawidłowych danych rejestracyjnych zostanie utworzone nowe konto, a użytkownik zostanie przekierowany na stronę logowania.

To login use below passwords:

- Normal user account. user name: **user**, password: **password**
- Manager account. user name: **manager**, password: **password**
- Administrator account. user name: **admin**, password: **password**

Account for user with name: ziggy has been created. Please log in!

User name

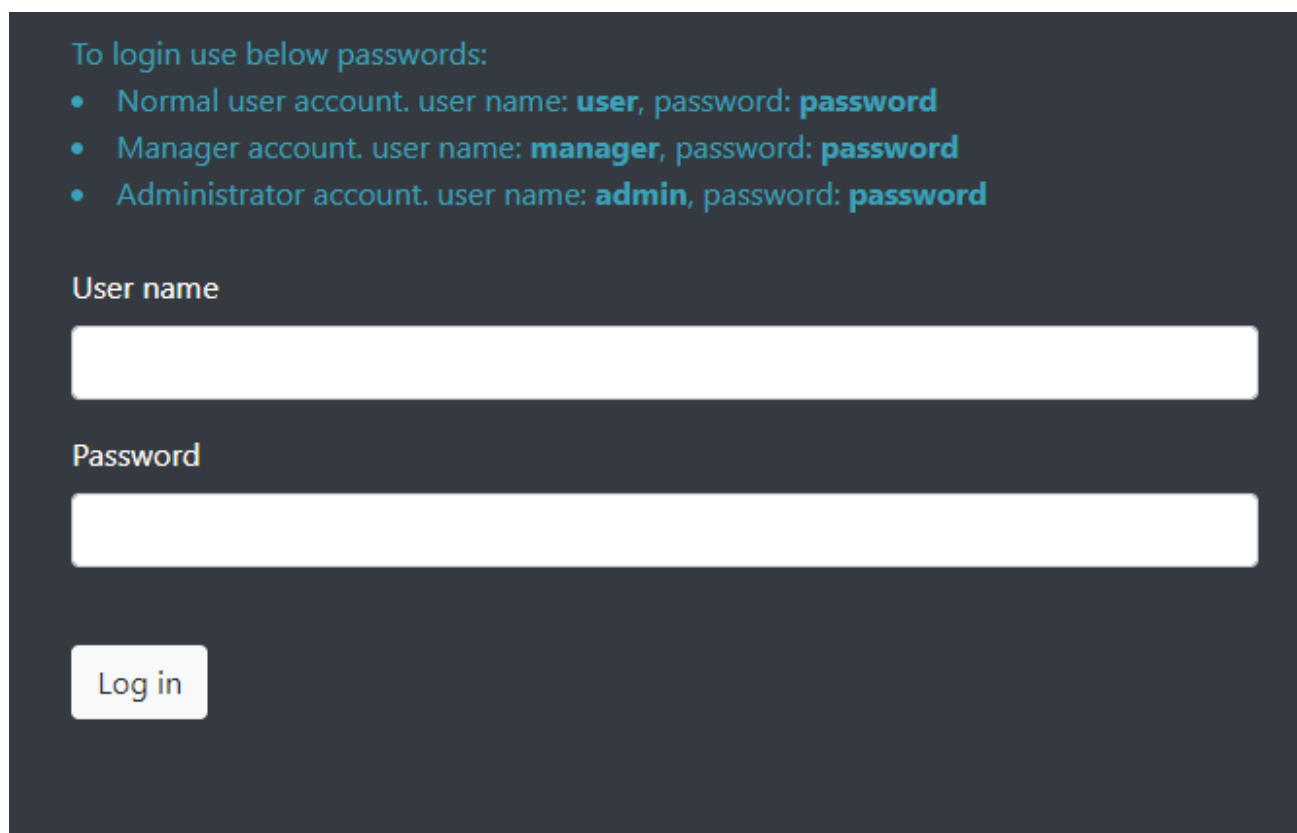
Password

Log in

© 2022

17.2. Logowanie do systemu

W celu zalogowania się do aplikacji należy wejść na główną stronę aplikacji, a następnie kliknąć zakładkę `Sign In` w głównym menu, w celu przekierowania do formularza logowania. Następnie należy wpisać prawidłowy login oraz hasło.



The screenshot shows a dark-themed login interface. At the top, it says "To login use below passwords:" followed by a bulleted list of three account types: "Normal user account. user name: **user**, password: **password**", "Manager account. user name: **manager**, password: **password**", and "Administrator account. user name: **admin**, password: **password**". Below this list are two input fields: "User name" and "Password", both with white text on a dark background. At the bottom left is a "Log in" button with white text on a dark background.

To login use below passwords:

- Normal user account. user name: **user**, password: **password**
- Manager account. user name: **manager**, password: **password**
- Administrator account. user name: **admin**, password: **password**

User name

Password

Log in

W przypadku wpisania błędnych danych użytkownik nie zostanie zalogowany, a na ekranie zostanie wyświetlony stosowny komunikat.

To login use below passwords:

- Normal user account. user name: **user**, password: **password**
- Manager account. user name: **manager**, password: **password**
- Administrator account. user name: **admin**, password: **password**

Invalid username and password.

User name

Password

Log in

Uwaga: W wersji MVP na ekranie logowania znajdują się informacje o danych do logowania do kont dla wszystkich typów użytkowników.

17.3. Wyszukiwanie dostępnych pojazdów

Aby wyświetlić ekran wyszukiwania samochodów, należy kliknąć zakładkę `Cars` w górnym menu [1]. Wstępnie zostaną wyświetlone wszystkie samochody. Aby znaleźć samochód dostępny w danym terminie, należy wprowadzić dwie daty `dateFrom` oraz `dateTo` [2]. Po wpisaniu dat wyświetlone zostaną wszystkie samochody dostępne w danym terminie. Aby zawęzić wyszukiwania można użyć dodatkowych filtrów [3] w celu wprowadzenia marki oraz modelu samochodu. Na ekranie wyświetlane jest pierwsze 5 znalezionych pojazdów, aby wyświetlić pozostałe wyniki, należy przejść do kolejnych stron wyszukiwania, w tym celu należy kliknąć numer strony znajdujący się pod wynikami wyszukiwania.

The screenshot shows the 'Car Rental App' interface. At the top, there is a navigation bar with the title '|| Car Rental App ||' and several menu items: 'Home', 'Cars' (highlighted with a red box and labeled '1'), 'Reservations', 'Messages' (with a dropdown arrow), and 'Logged user: manager' (with a dropdown arrow). Below the navigation bar, the heading 'Available cars:' is displayed. Under this heading, there are two date input fields: 'Date from: dd.mm.rrrr' and 'Date to: dd.mm.rrrr', both with calendar icons. These fields are grouped by a red box and labeled '2'. Below the date fields is a table of available cars. The table has columns for 'Id', 'Manufacturer', 'Model', 'Cost per day', and four action buttons: 'Book', 'View Details', 'Edit', and 'Delete'. The first row of the table is highlighted with a red box and labeled '3'. Below the table, there are two pagination buttons: '1' and '2'.

Id	Manufacturer	Model	Cost per day				
Filters:	<input type="text"/>	<input type="text"/>					
1	Opel	Astra	130,00 PLN	Book	View Details	Edit	Delete
2	Opel	Corsa	95,00 PLN	Book	View Details	Edit	Delete
3	Fiat	500	95,00 PLN	Book	View Details	Edit	Delete
4	Fiat	Tipo	110,00 PLN	Book	View Details	Edit	Delete
5	Kia	Sorento	230,00 PLN	Book	View Details	Edit	Delete

Uwaga: W wersji MVP aplikacji nie zaimplementowano sortowania ani możliwości wyboru ilości stron do wyświetlenia na ekranie logowania. Funkcje te znajdą się w pełnej wersji aplikacji.

17.4. Składanie rezerwacji

W celu złożenia rezerwacji musimy posiadać konto w serwisie. Po rejestracji lub zalogowaniu (kroki 19.1 i 19.2) należy podobnie jak w kroku 19.3, kliknąć zakładkę `cars` [1] dzięki której uzyskamy dostęp do pełnej listy pojazdów. W celu zarezerwowania pojazdu należy najpierw określić termin, w jakim chcielibyśmy dokonać rezerwacji [2]. Następnie wybrać dostępny w tym terminie pojazd naciskając przycisk `Book` [3], dzięki któremu uzyskamy dostęp do podglądu potwierdzenia naszej rezerwacji.

|| Car Rental App || Home **Cars** Reservations Send Message Logged user: user 1

Available cars:

Date from: dd.mm.rrrr Date to: dd.mm.rrrr 2

Id	Manufacturer	Model	Cost per day		
Filters:	<input type="text"/>	<input type="text"/>			
1	Opel	Astra	130,00 PLN	Book 3	View Details
2	Opel	Corsa	95,00 PLN	Book	View Details
3	Fiat	500	95,00 PLN	Book	View Details
4	Fiat	Tipo	110,00 PLN	Book	View Details
5	Kia	Sorento	230,00 PLN	Book	View Details

1 2

Po dokładnym zapoznaniu się z danymi rezerwacji należy wcisnąć przycisk submit [1] potwierdzający złożenie rezerwacji w określonym terminie i po określonej cenie lub przycisk Go Back który pozwala na rezygnację z rezerwacji i powrót do okna listy dostępnych pojazdów [2].

Confirm reservation:

User

Id

Manufacturer

Model

Date From

Date To

Total cost [PLN]

1 **Submit** **Go Back** 2

17.5 Przeglądanie rezerwacji oraz usuwanie rezerwacji

Aby wyświetlić listę złożonych rezerwacji, należy po zalogowaniu lub rejestracji wejść w zakładkę `Reservations` [1]. Wyświetlona zostanie lista rezerwacji złożonych na tym koncie, nazwa pojazdu, czas rezerwacji oraz jej koszt. W celu zobaczenia dokładnych danych pojedynczej rezerwacji należy nacisnąć przycisk `View Details` [2]. Program pozwala również na anulowanie zamówienia za pomocą przycisku `Cancel Reservation` [3].

Id	Car	Date from	Date to	Total cost		
1	Opel Corsa	2022-09-15	2022-09-18	250,00 PLN	View Details	Cancel Reservation
2	Opel Corsa	2022-07-01	2022-07-18	1.234,00 PLN	View Details	Cancel Reservation
3	Fiat 500	2022-08-01	2022-08-18	1.234,00 PLN	View Details	Cancel Reservation
4	Fiat Tipo	2022-09-01	2022-09-18	1.800,00 PLN	View Details	Cancel Reservation
5	Opel Astra	2022-10-10	2022-10-18	800,00 PLN	View Details	Cancel Reservation

Po naciśnięciu przycisku `View Details` uzyskamy dostęp do panelu pozwalającego na podgląd danych dotyczących tylko jednego wybranego zamówienia. Panel ten pozwala nam również na anulowanie zlecenia `Cancel Reservation` [1] oraz wygenerowanie pliku PDF z wszystkimi danymi tego zlecenia `Generate PDF` [1].

Uwaga: Panel zawiera również funkcję `Edit` [3] pozwalającą na edytowanie zlecenia. Funkcja ta jest dostępna tylko dla Zarządcy lub *Administratora*.

Reservation details:

Purchaser:
James Smith

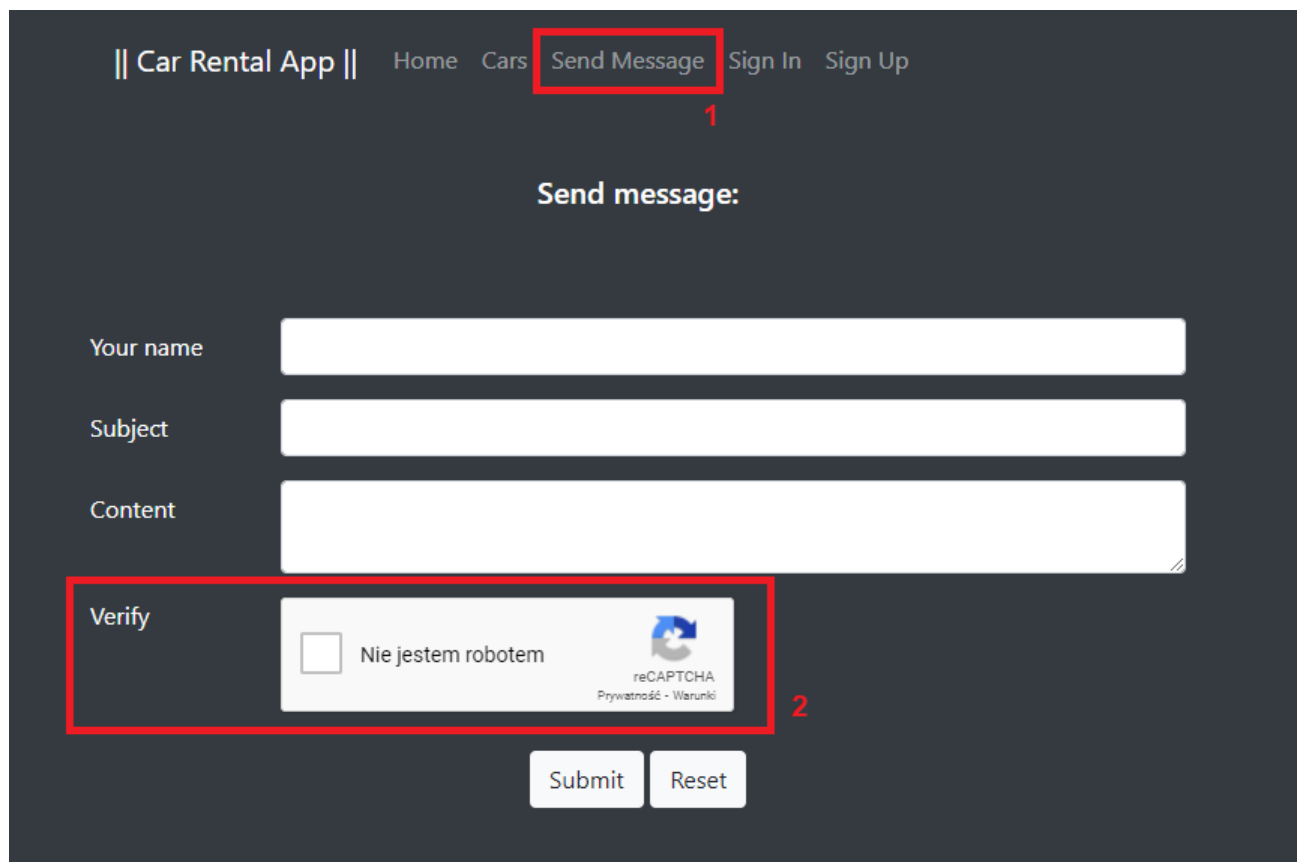
Reservation no: 1

Reservation Date	Car	Rent period	Total Cost
2022-01-15	Opel Corsa	from: 2022-09-15 to: 2022-09-18	250.00

Edit Cancel Reservation Generate PDF

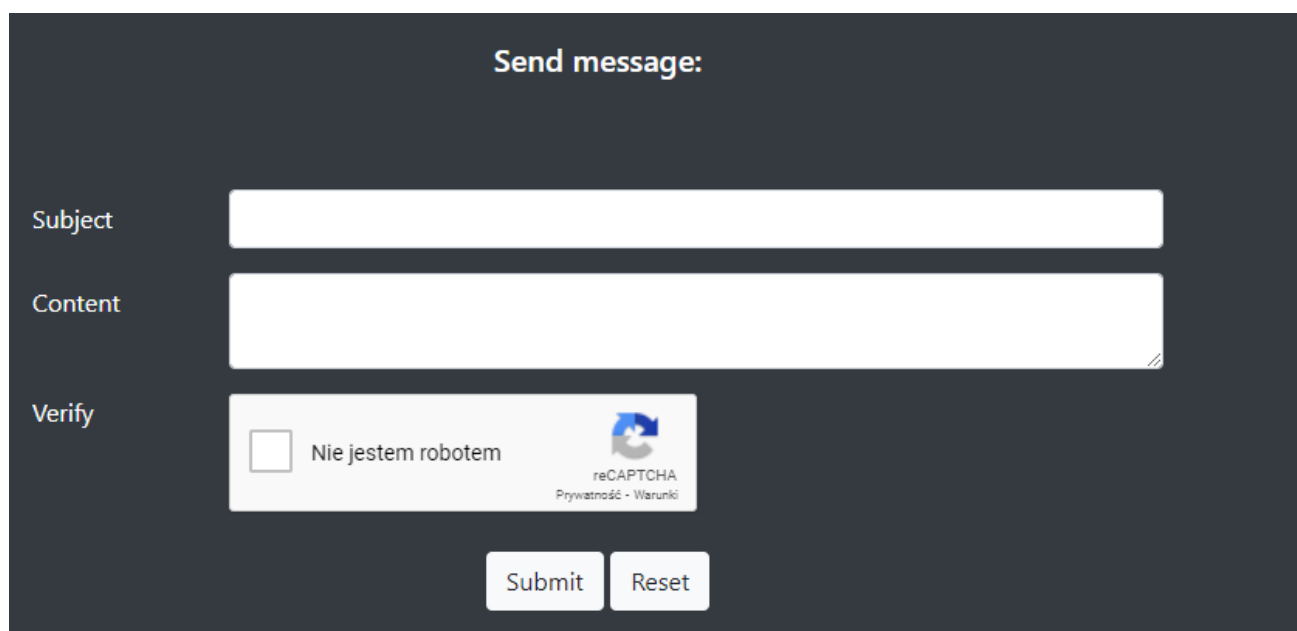
17.6 Wysyłanie wiadomości

W celu wysłania wiadomości do administracji **jako użytkownik niezalogowany** należy nacisnąć w panel **Send Message** [1], a następnie uzupełnić formularz zgodnie z tytułami pól. Po uzupełnieniu formularza należy potwierdzić pole captcha w **verify** zabezpieczające przed spamem [2].



The screenshot shows a dark-themed web interface for a 'Car Rental App'. At the top, there is a navigation bar with links: 'Home', 'Cars', 'Send Message' (highlighted with a red box and labeled '1'), 'Sign In', and 'Sign Up'. Below the navigation bar, the heading 'Send message:' is centered. The form consists of three input fields: 'Your name', 'Subject', and 'Content'. Below these fields is a 'Verify' section (highlighted with a red box and labeled '2') containing a checkbox labeled 'Nie jestem robotem' and a reCAPTCHA logo with the text 'reCAPTCHA Prywatność - Warunki'. At the bottom of the form are two buttons: 'Submit' and 'Reset'.

W razie próby wysłania wiadomości jako **zalogowany użytkownik**, panel ten wygląda trochę inaczej. Pole z imieniem jest uzupełniane automatycznie według loginu użytkownika.



The screenshot shows the same 'Send message:' form, but for a logged-in user. The 'Your name' field is missing, and the 'Subject' field is the first input field. The 'Content' field is the second input field. The 'Verify' section (checkbox and reCAPTCHA) and the 'Submit' and 'Reset' buttons remain at the bottom.

17.7. Funkcje dostępne dla Zarządcę

Wszystkie funkcje przedstawione w tym rozdziale jest niedostępne dla zwykłego użytkownika. W celu ich obsługi przez osobę zarządzającą została stworzona rola zarządcy.

17.7.1 Edycja pojazdów

W celu edycji danych pojazdu trzeba udać się do panelu cars [1], a następnie kliknąć przycisk edit [2] na wybranym pojeździe, który chcemy edytować.

The screenshot shows the 'Car Rental App' interface. At the top, there is a navigation bar with the following items: '|| Car Rental App ||', 'Home', 'Cars' (highlighted with a red box and labeled '1'), 'Reservations', 'Messages', and 'Logged user: manager'. Below the navigation bar, the section 'Available cars:' is visible. It includes two date input fields: 'Date from: dd.mm.rrrr' and 'Date to: dd.mm.rrrr'. Below these fields is a table of available cars. The table has columns: 'Id', 'Manufacturer', 'Model', 'Cost per day', 'Book', 'View Details', 'Edit', and 'Delete'. The 'Edit' button for the first car (Opel Astra) is highlighted with a red box and labeled '2'. At the bottom of the table, there are two pagination buttons: '1' and '2'. Below the table, there is an 'Add New Car' button.

Id	Manufacturer	Model	Cost per day	Book	View Details	Edit	Delete
1	Opel	Astra	130,00 PLN	Book	View Details	Edit	Delete
2	Opel	Corsa	95,00 PLN	Book	View Details	Edit	Delete
3	Fiat	500	95,00 PLN	Book	View Details	Edit	Delete
4	Fiat	Tipo	110,00 PLN	Book	View Details	Edit	Delete
5	Kia	Sorento	230,00 PLN	Book	View Details	Edit	Delete

Po wykonaniu tych kroków uzyskujemy widok edycji pojazdu. Aby edytować pojazd, należy uzupełnić formularz, a następnie zatwierdzić go przyciskiem submit [1]. W przypadku chęci powrotu do danych początkowych na należy nacisnąć przycisk Reset [2].

Edit car:

Car id	1
Registration plate	ST 2312U
Manufacturer	Opel
Model	Astra
Fuel type	gasoline
Car type	passenger car
Engine capacity	1478
Passenger capacity	5
Description	Samochód osobowy klasy kompakt
Cost per day PLN	130,00

1

2

Submit

Reset

17.7.2 Usuwanie pojazdów

W celu usunięcia pojazdu należy wejść w panel cars [1], a następnie wcisnąć przycisk Delete na wybranym pojeździe [2].

|| Car Rental App || Home **Cars** Reservations Messages Logged user: manager

1

Available cars:

Date from: dd.mm.rrrr Date to: dd.mm.rrrr

Id	Manufacturer	Model	Cost per day				
Filters:							2
1	Opel	Astra	130,00 PLN	Book	View Details	Edit	Delete
2	Opel	Corsa	95,00 PLN	Book	View Details	Edit	Delete
3	Fiat	500	95,00 PLN	Book	View Details	Edit	Delete
4	Fiat	Tipo	110,00 PLN	Book	View Details	Edit	Delete
5	Kia	Sorento	230,00 PLN	Book	View Details	Edit	Delete

1 2

Add New Car

17.7.3 Przeglądanie rezerwacji wszystkich użytkowników

Aby wyświetlić listę wszystkich rezerwacji użytkowników, należy wejść w panel Reservations [1]. Na ekranie wyświetli się lista zarezerwowanych samochodów wraz z loginem użytkownika, który zarezerwował dany pojazd.

|| Car Rental App || Home Cars **Reservations** Messages Logged user: manager

1

Reservation list:

Id	User Name	Car	Date from	Date to	Total cost			
Filters:								
3	user	Fiat 500	2022-08-01	2022-08-18	1.234,00 PLN	View Details	Edit	Delete
4	user	Fiat Tipo	2022-09-01	2022-09-18	1.800,00 PLN	View Details	Edit	Delete
5	user	Opel Astra	2022-10-10	2022-10-18	800,00 PLN	View Details	Edit	Delete
6	user	Fiat Tipo	2022-05-10	2022-05-18	800,00 PLN	View Details	Edit	Delete
7	manager	Opel Astra	2022-05-18	2022-05-22	520,00 PLN	View Details	Edit	Delete

1

17.7.4 Edycja rezerwacji

W celu edycji danych rezerwacji trzeba udać się do panelu `Reservations` [1], a następnie kliknąć przycisk `Edit` [2] na wybranej rezerwacji,, którą chcemy edytować.

|| Car Rental App || Home Cars **Reservations** Messages Logged user: manager

1

Reservation list:

Id	User Name	Car	Date from	Date to	Total cost			
Filters:	<input type="text"/>							2
3	user	Fiat 500	2022-08-01	2022-08-18	1.234,00 PLN	View Details	Edit	Delete
4	user	Fiat Tipo	2022-09-01	2022-09-18	1.800,00 PLN	View Details	Edit	Delete
5	user	Opel Astra	2022-10-10	2022-10-18	800,00 PLN	View Details	Edit	Delete
6	user	Fiat Tipo	2022-05-10	2022-05-18	800,00 PLN	View Details	Edit	Delete
7	manager	Opel Astra	2022-05-18	2022-05-22	520,00 PLN	View Details	Edit	Delete

1

Uzyskujemy tym sposobem widok edycji rezerwacji. W celu edycji pojazdu należy uzupełnić formularz, a następnie zatwierdzić go przyciskiem `Submit` [1]. W przypadku chęci powrotu do danych początkowych na należy nacisnąć przycisk `Reset` [2].

Id: 3

User: user

Car: 500

Date From: 01.08.2022

Date From: 18.08.2022

Total cost [PLN]: 1234.00

Rented: ☐

1 2

Submit Reset

17.7.5 Usuwanie rezerwacji

W celu usunięcia rezerwacji należy wejść w panel `Reservations` [1], a następnie wcisnąć przycisk `Delete` na wybranej rezerwacji [2].

|| Car Rental App || Home Cars **Reservations** Messages Logged user: manager

1

Reservation list:

Id	User Name	Car	Date from	Date to	Total cost			
Filters:	<input type="text"/>							2
3	user	Fiat 500	2022-08-01	2022-08-18	1.234,00 PLN	View Details	Edit	Delete
4	user	Fiat Tipo	2022-09-01	2022-09-18	1.800,00 PLN	View Details	Edit	Delete
5	user	Opel Astra	2022-10-10	2022-10-18	800,00 PLN	View Details	Edit	Delete
6	user	Fiat Tipo	2022-05-10	2022-05-18	800,00 PLN	View Details	Edit	Delete
7	manager	Opel Astra	2022-05-18	2022-05-22	520,00 PLN	View Details	Edit	Delete

1

17.7.6 Odczytywanie i zarządzanie wiadomościami

Zarządca ma możliwość odczytywania wiadomości od użytkowników dotyczących wynajmu. W celu dostania się do panelu wiadomości należy nacisnąć Messages [1], a następnie z rozsuwanej listy wybrać View Messages [2]. W przypadku chęci przeczytania pełnej treści wiadomości należy nacisnąć przycisk view [3]*, a w przypadku chęci usunięcia wiadomości należy nacisnąć przycisk Delete [4].

Uwaga: W wersji MVP podgląd wiadomości nie został zaimplementowany.

|| Car Rental App || Home Cars Reservations **Messages** Logged user: manager

1

Messages

2

Id	Sent date	From	Subject	Message		
1	2022-04-10	user	Test message	This is just test message	3 View	4 Delete
2	2022-04-11	user	Test message 2	This is just test message	View	Delete

1

17.8 Funkcje dostępne dla Administratora

Administrator to specjalny użytkownik zawierający oprócz wszystkich funkcji poniżej dostęp dwóch specjalnych funkcji przeznaczonych tylko dla niego.

17.8.1 Zarządzanie użytkownikami

W celu dostania się do panelu zarządzania użytkownikami należy nacisnąć przycisk `Settings` [1], a następnie z rozsuwanej listy `User Administration` [2]. Panel ten pozwala nam na dodawanie (przycisk `Add User` [3], edycję (przycisk `Edit` [4]) oraz usuwanie użytkowników (przycisk `Delete` [5]).

UserName	First Name	Last Name	Role	Edit	Delete
admin	John	Doe	ADMIN	Edit	Delete
user	James	Smith	USER	Edit	Delete
manager	James	Smith	MANAGER	Edit	Delete

17.8.1.1 Dodawanie użytkownika

Po wciśnięciu przycisku `Add User` uzyskujemy dostęp do formularza, w którym dodajemy dane nowego użytkownika oraz przypisujemy mu określoną rolę.

Wszelkie zmiany należy zatwierdzić przyciskiem `Submit` [1], natomiast w przypadku chęci wyczyszczenia formularza należy użyć przycisku `Reset` [2].

Add User:

Login:

First Name:

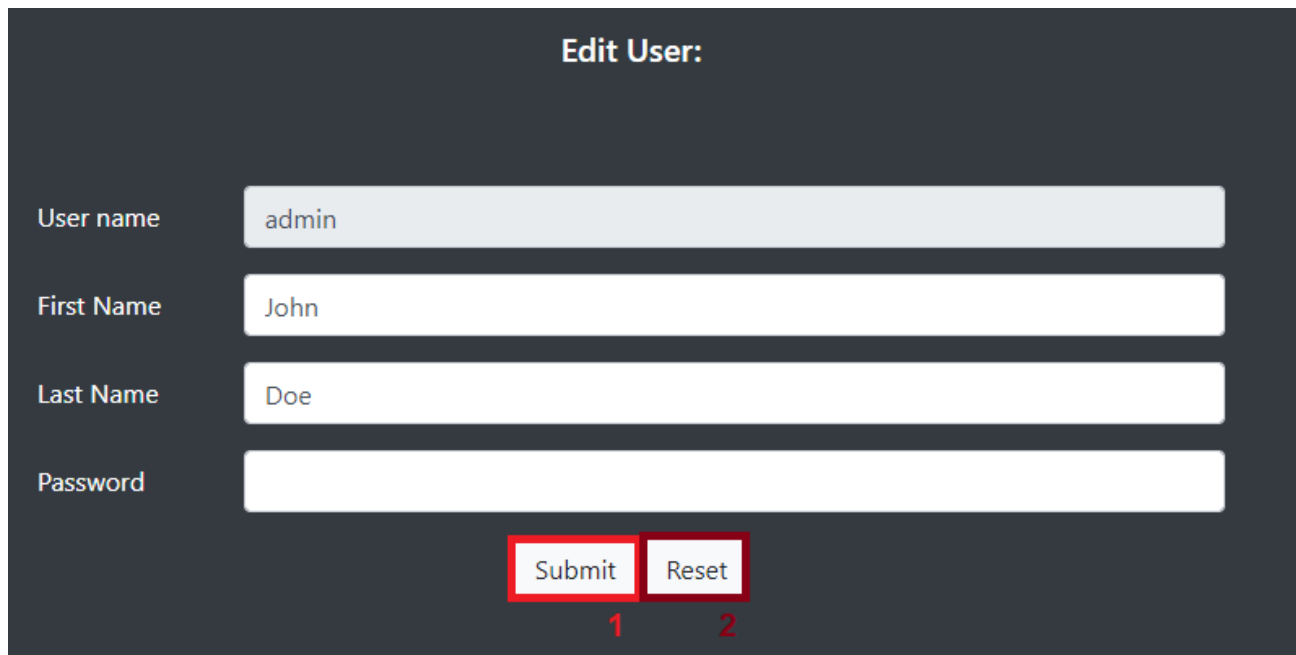
Last Name:

Role:

Password:

17.8.1.2 Edycja użytkownika

Po użyciu przycisku `Edit` na wybranym użytkowniku wyświetla się formularz edycji, w którym możemy dokonać potrzebnych nam zmian. Każdą zmianę należy zatwierdzić przyciskiem `Submit` [1], natomiast w przypadku chęci wyczyszczenia formularza należy użyć przycisku `Reset` [2].



Edit User:

User name: admin

First Name: John

Last Name: Doe

Password:

Submit (1) Reset (2)

17.8.2 Ustawienia administracyjne

UWAGA: Panel ustawień administracyjnych w wersji MVP jest wyłącznie pokazowy, wszelkie funkcje nie zostały jeszcze zaimplementowane.

Aby dostać się do ustawień administracyjnych trzeba wejść w panel `Settings` [1], a następnie z rozwijanej listy wybrać `Administration Settings`. Panel ten pozwala nam na zmianę danych dotyczących firmy [3] oraz zmianę waluty obsługiwanej na stronie [4]. Wszelkie zmiany zatwierdzone są przyciskiem `Submit` [5], a w celu przywrócenia poprzednich danych należy nacisnąć przycisk `Reset` [6].

|| Car Rental App ||

HomeCarsReservationsMessages

Settings

Logged user: admin

Administration Settings

User Administration

Application Settings (under construction)

Company data

Company NameTwoja Firma

StreetTwoja Ulica

Property No.10

Postal code43-100

CityMiasto

CountryPolska

Phone Number797343234

Currency Settings

CurrencyPLN

Current exchange rate (from NBP Web API):

PLN	EUR	GBP	USD
1	4.6528	5.4797	4.4279

Other Settings

under construction && additional settings will be added here later on

SubmitReset